UNIVERSIDAD COMPLUTENSE DE MADRID
FACULTAD DE INFORMÁTICA
Departamento de Arquitectura de Computadores y Automática

TESIS DOCTORAL

Optimización de rendimiento y consumo energético del subsistema de memoria mediante nuevas técnicas basadas en metaheurísticas

MEMORIA PARA OPTAR AL GRADO DE DOCTOR

PRESENTADA POR

Josefa Díaz Álvarez

Directores

José Luis Risco Martín
José Manuel Colmenar Verdugo

Madrid, 2018

UNIVERSIDAD COMPLUTENSE MADRID

FACULTAD DE INFORMÁTICA

Optimización de rendimiento y consumo energético del subsistema de memoria mediante nuevas técnicas basadas en metaheurísticas

TESIS DOCTORAL

Autora:
Dña. Josefa Díaz Álvarez

Directores:
Dr. José Luis Risco Martín
Dr. José Manuel Colmenar Verdugo

Madrid 2016

Optimización de rendimiento y consumo energético del subsistema de memoria mediante nuevas técnicas basadas en metaheurísticas

Memoria presentada por Josefa Díaz Álvarez para optar al grado de Doctor por la Universidad Complutense de Madrid, realizada bajo la dirección de Dr. José Luis Risco Martín y Dr. José Manuel Colmenar Verdugo (Departamento de Arquitectura de los Computadores y Automática. Universidad Complutense de Madrid.)

Este trabajo ha sido posible gracias a la financiación recibida por el Ministerio de Economía y Competitividad a través de los proyectos de investigación TIN2014-54806-R, TIN2014-56494-C4-2-P y TIN2014-56494-C4-2-P.

Agradecimientos

Quiero agradecer el interés con que mis directores José Luis y Chema han seguido mi trabajo, sus consejos, sus advertencias, pero sobre todo por la humanidad y el cariño que siempre he recibido. Muchas gracias porque siempre me he sentido acompañada y que formaba parte de un equipo. He aprendido mucho y me queda mucho por aprender.

Gracias a Paco por guiarme en este camino, a Iñaqui, Juan, Oscar y al personal de la UCM. Cuando he venido me habéis acogido como uno más. Mi más sincero agradecimiento a todos ellos y a mis compañeros que han aguantado mi mal humor, mis fatigas, mis quejas, mis malos momentos y mis buenos momentos.

Gracias a mi familia, a los que están conmigo y a los que estoy segura que me sienten. Hoy mi padre hubiera sido muy feliz. Sin embargo, esta tesis está dedicada a las tres personas más importantes en mi vida mis hijas y mi marido. A mis hijas a las que va dirigido mi pensamiento cada mañana al levantarme y que espero que aprendan lo duro que es conseguir las metas que uno se pone. Pero, sobre todo a ti, Antonio, durante estos años has tenido que aprender a hacer de todo porque yo no estaba en muchas ocasiones. Sobre todo, has aprendido a cocinar como nadie, mis ausencias han tenido más puntos positivos que negativos, ¿lo ves?.

Gracias porque sin vosotros no se si hubiera llegado hasta aquí.

Índice general

Lista de Acrónimos

3D-IC Three Dimensional Integration Circuit. Circuito Integrado en 3 Dimensiones.

ACO Ant Colony Optimization. Optimización de Colonia de Hormigas.

ADR Adaptive Drop Rate. Tasa de Caida Adaptativa.

ASIP Application-Specific Instruction-Set Processor. Procesador con conjunto de Instrucciones de Aplicación Específica.

BNF Backus Naur Form. Notación Backus Naur.

CAD Computer Aided Design. Diseño Asistido por Ordenador.

CMP Chip Multiprocessor. Chip Multiprocesador.

CMT Coarse-grained Multithreading. Multitarea de Grano Grueso.

CPI Cycles Per Instruction. Ciclos por Instrucción.

CPU Central Processing Unit. Unidad Central de Proceso.

DDT Dynamic Data Type. Tipo de Datos Dinámico.

DE Differential Evolution. Evolución Diferencial.

DMM Dynamic Memory Management. Gestor de Memoria Dinámica.

DRAM Dynamic Random Access Memory. Memoria de Acceso Aleatorio Dinámico.

DSP Digital Signal Processing. Procesamiento Digital de Señal.

DVS Dynamic Voltage Scale. Escalado Dinámico de Voltaje.

EA Evolutionary Algorithm. Algoritmo Evolutivo.

EC Evolutionary Computation. Computación Evolutiva.

EP Evolutionary Programing. Programación Evolutiva.

EPIC Explicitly Parallel Instruction Computing. Procesamiento de Instrucciones Explicitamente Paralelas.

ES Evolutionary Estrategies. Estrategias Evolutivas.

FMT Fine-grained Multithreading. Multitarea de Grano Fino.

GAs Genetic Algorithms. Algoritmos Genéticos.

GE Grammatical Evolution. Evolución Gramatical.

GHz GigaHerz. Gigaherzio.

GP Genetic Programming. Programación Genética.

GUI Graphical User Interface. Interface Gráfica de Usuario.

IC Integrated Circuit. Circuito Integrado.

ILP Instruction Level Paralelism. Paralelismo a Nivel del Instrucción.

ILS Iterated Local Search. Búsqueda Local Iterativa.

LIFO Last Input First Output. Último en entrar, primero en salir.

LISP LISt Processing. Procesamiento de Listas.

MHz MegaHerz. Megaherzio.

MOEA Muti-Objective Evolutionary Algorithm. Algoritmo Evolutivo Multi-objetivo.

MOP Multi-objective Optimization Problem. Problema de Optimización Multi-objetivo.

MPSoC Multiprocessor System-on-Chip. Multiprocesador on-chip.

NEMM Nanoelectromechanical Memory. Memoria Electromecánica.

PCRAM Phase Change Random Access Memory. Memoria de Acceso Aleatorio con Cambio de Fase.

POF Pareto Optimal Front. Frente Óptimo de Pareto.

POS Pareto Optimal Set. Conjunto Óptimo de Pareto.

RISC Reduced Instruction Set Computing. Procesamiento de un Conjunto Reducido de Instrucciones.

ROI Region Of Interest. Región de Interés.

RRAM Resistive Random Access Memory. Memoria de Acceso Aleatorio Resistiva.

RTS Real Time System. Sistema en Tiempo Real.

SA Simulated Annealing. Enfriamiento Simulado.

SMT Simultaneous Multithreading. Multitarea Simultánea.

SoC System-on-Chip.

SRAM Static Random Access Memory. Memoria de Acceso Aleatorio Estático.

SSD Solid State Disk. Disco de Estado Sólido.

STTRAM Spin Torque Transfer Random Access Memory.

TLP Thread Level Paralelism. Paralelismo a Nivel de Tarea.

VLIW Very Long Instruction Word. Palabra de Instrucción muy larga.

Abstract

Nowadays, embedded system design for mobile devices has evolved during last years adapting its characteristic to the users needs. However, this sort of devices is battery-powered with a limited capacity because of design constraints like the size and the device weight where it is included.

Besides, current mobile devices often execute multimedia applications, which generally require high performance and cause high power consumption. In this context, the degradation of the devices due to the intensive use, the thermal issues and the high power consumption affects to systems and can cause irreversible damages in devices.

From this perspective, one of the most important challenges that systems designers face is the need of increasing performance while reducing energy consumption. Particularly, the memory subsystem, formed by the processor registers, the cache memory and the main memory, is one of the system components with major influence on performance and power consumption. In the case of multimedia applications, the memory subsystem receives a great workload, because of the inherent characteristics of this kind of applications.

The objectives of this thesis encompass the optimization of the memory subsystem at all its levels: processor registers, cache memory and main memory, in this case through the dynamic memory management.

The first objective is the thermal optimization of the register file taking the VLIW and specially the ARM architectures as target systems. In the optimization of the register file, a configuration will be obtained which, being physically viable, will have the lowest thermal impact.

The second objective consists in the optimization of both the performance and the energy consumption in the cache memory subsystem. To this end, ARM is selected as the target architecture. The idea is to explore the different cache configurations, selecting the optimal values for parameters that define a cache configuration, and finding those cache configurations which increase the performance and reduce the power consumption.

The third objective addresses the optimization of dynamic memory management in the main memory. The dynamic memory manager is a crucial component, and is responsible for making the allocation and deallocation of dynamic memory. Therefore, it has a great influence on the behavior of applications in the hardware platforms. For the main memory, the aim is to design custom dynamic memory managers which minimize memory usage and maximize performance on each application under study.

To achieve these goals three optimization frameworks are proposed, where multimedia applications are used. In all these frameworks, a first stage of static profiling is made on each target application. Then, different metaheuristic techniques are applied, which allow to address the optimization of every level in the memory system, as previously described.

The thermal optimization of the registers file begins with a thermal analysis according to the thermal model in steady-state defined by Brooks. After the analysis phase, a multi-objective optimization process is proposed, where each solution is a distribution of accesses to registers on the registers file.

In the cache memory optimization, the performance and energy models provided by Janapsatya are applied and an optimization process is proposed based on three different evolutionary algorithms. In this way, it is possible to identify the best cache configuration according to the running applications, with different starting conditions. The first algorithm, which applied Grammatical Evolution, addresses the optimization process for each individual application, by means of a weighted function of the objectives under study, The second algorithm performs the same

individual optimization, but it tackles the problem applying the multi-objective algorithm NSGA-II. The third approach addresses the optimization considering all the applications under study together, applying a Differential Evolution algorithm.

Regarding the dynamic memory management, the Grammatical Evolution is used to design custom dynamic memory managers tailored to the application under study. The grammar on this optimization is designed based on the features of the target application, which are taken from the application trace. The optimization algorithm evaluates each one of the dynamic memory managers generated by the algorithm, and the execution time and memory usage are normalized with respect to Kingsley and Lea managers, given that they are, respectively, the fastest and most efficient ones according to the literature.

The experimental results show savings in the thermal impact of the registers file for all the analized applications, reaching values higher above 10 % for some applications in terms of the maximum average temperature and the maximum temperature decrease.

Regarding the cache memory, the optimization process obtains cache configurations, which improve more than 30 % and 70 % the behavior in performance and power consumption, respectively.

In relation to the main memory, dynamic memory managers designed are presented as the best option for the whole set of the target applications. The overall improvement reaches up to 90,78 %.

In this thesis different metaheuristic techniques have been applied during the optimization process of the memory subsystem very common to the current mobile devices.

The results confirm that the memory subsystem optimization provides significant improvements, especially in the case of the cache memory and dynamic memory. Thus, in dedicated portable devices, it is crucial to perform an optimization process similar to the one described in this work.

The solutions space with this sort of optimizations is large, and becomes lar-

ger with the increase of the number of possible configurations supported by the target architecture. In addition, the evaluation of each configuration of a memory subsystem is really costly in terms of wall-clock time. Consequently, the use of metaheuristics in this scenario is therefore justified.

Finally, a methodology is provided to the system designer that facilitates decision-making in the design of the different levels of the memory hierarchy.

Resumen

Actualmente, todo dispositivo móvil dispone de un sistema empotrado cuyo diseño ha evolucionado en los últimos años, adaptando sus características a las necesidades de los usuarios. Sin embargo, este tipo de dispositivos tiene como fuente de alimentación una batería cuya capacidad está limitada por restricciones de diseño como el tamaño y el peso del dispositivo en que se incluye.

Por otra parte, los dispositivos móviles actuales ejecutan frecuentemente aplicaciones multimedia que, generalmente, requieren elevadas prestaciones en cuanto a rendimiento del dispositivo, provocando un alto consumo de energía. En este contexto, la degradación de los dispositivos debida al uso intensivo, los problemas térmicos y el alto consumo de energía afectan a los sistemas y pueden provocar daños irreversible en los dispositivos.

Desde este punto de vista, uno de los retos más importantes que se plantean los diseñadores de sistemas es la necesidad de incrementar el rendimiento a la vez que se reduce el consumo de energía. Más concretamente, el subsistema de memoria, compuesto por los registros de procesador, la memoria cache y la memoria principal, es uno de los componentes del sistema con mayor influencia tanto en el rendimiento como en el consumo de energía. En el caso de las aplicaciones multimedia, el subsistema de memoria recibe una gran carga de trabajo, debido a las características inherentes a este tipo de aplicaciones.

Los objetivos de esta tesis engloban la optimización del subsistema de memoria en todos sus niveles: registros del procesador, memoria cache y memoria principal (a través de la gestión de memoria dinámica).

El primer objetivo es la optimización térmica del banco de registros tomando

como modelo las características de las arquitecturas VLIW y, principalmente, ARM. En este caso, se obtendrá una configuración que, siendo viable físicamente, tenga el menor impacto térmico posible.

El segundo objetivo consiste en optimizar tanto el rendimiento como el consumo de energía en el subsistema de memoria cache. Para ello, se toma ARM como arquitectura de trabajo, y se explorarán las diferentes configuraciones de cache, seleccionando los valores óptimos del conjunto de parámetros que la definen. Así, se encontrarán aquellas configuraciones que incrementan el rendimiento y reducen el consumo de energía.

El tercer objetivo aborda la optimización de la gestión de memoria dinámica en la memoria principal. El gestor de memoria dinámica es un componente fundamental que se encarga de realizar la asignación y liberación dinámica de memoria. Por ello tiene una gran influencia en el comportamiento de las aplicaciones y el rendimiento de las plataformas hardware. Para la memoria principal se buscará diseñar gestores de memoria dinámica que minimicen el uso de la memoria y maximicen el rendimiento.

Para conseguir estos objetivos se proponen tres marcos de optimización donde se utilizarán aplicaciones multimedia. En estos marcos se optimización se realiza, en primer lugar, un perfilado estático de las aplicaciones. A continuación, se aplican diferentes técnicas metaheurísticas que permiten abordar la optimización de cada una de las partes del sistema de memoria mencionadas en la enumeración de objetivos.

En la optimización térmica del banco de registros se comienza realizando un análisis térmico en estado estacionario según el modelo térmico definido por Brooks. Después de la fase de análisis, se propone un proceso de optimización multi-objetivo, donde cada solución representa una distribución de los registros dentro del banco de registros.

En la optimización de la memoria cache se utiliza el modelo de rendimiento y energía propuesto por Janapsatya, sobre él se propone un proceso de optimización

basado en tres diferentes algoritmos evolutivos. De esta manera se consigue identificar la mejor configuración de cache de acuerdo a las aplicaciones en ejecución en diferentes condiciones de partida. El primer algoritmo, que aplica Gramáticas Evolutivas, aborda el proceso de optimización para cada aplicación de manera individual, pero a través de una función ponderada de los objetivos en estudio. El segundo algoritmo plantea la misma optimización individual, pero separando los objetivos a través del algoritmo multi-objetivo NSGA-II. El tercer enfoque aborda el proceso de optimización considerando de forma conjunta seis de las doce aplicaciones en estudio y aplicando un algoritmo de Evolución Diferencial.

En la optimización de la gestión de memoria dinámica se utilizan Gramáticas Evolutivas para diseñar gestores a medida de cada aplicación bajo estudio. La gramática se diseña en base a las características de la aplicación objetivo que son extraídas de la traza de la aplicación. El algoritmo de optimización evalúa cada gestor de memoria dinámica generado mediante la función objetivo, donde tiempo de ejecución y uso de memoria se normalizan con respecto a los gestores Kingsley y Lea, considerados como el más rápido y el más eficiente según la literatura.

Los resultados muestran una reducción del impacto térmico del banco de registros en todas las aplicaciones analizadas, alcanzando en algunas valores superiores al 10 %, respecto a la temperatura promedio y máxima. Respecto a la memoria cache, se encuentran configuraciones que mejoran más del 30 % y 70 % en rendimiento y consumo de energía, respectivamente. Respecto a la memoria principal, los gestores de memoria dinámica diseñados se presentan como la mejor opción para el conjunto de aplicaciones objetivo, con una mejora promedio hasta el 90, 78 %.

En esta tesis, por tanto, se han aplicado diferentes técnicas metaheurísticas en el proceso de optimización del subsistema de memoria común a los dispositivos móviles actuales.

Los resultados confirman que la optimización del subsistema de memoria aporta mejoras muy significativas, especialmente en el caso de la memoria cache y de la memoria dinámica. Por ello, resulta fundamental realizar un proceso similar al

descrito en este trabajo en sistemas dedicados a la ejecución de aplicaciones muy concretas.

El espacio de soluciones en este tipo de optimizaciones es grande, y aumenta cuantas más posibles configuraciones admita una arquitectura destino. Además, la evaluación de cada una de las configuraciones en cualquier parte de la memoria es muy costosa en términos de tiempo. En consecuencia, la utilización de metaheurísticas en este escenario es fundamental.

Finalmente, se proporciona al diseñador del sistema una metodología que facilita la toma de decisiones en los niveles de la jerarquía de memoria abordados.

Capítulo 1

Introducción

1.1. Motivación

En las últimas décadas se han producido logros considerables en el campo tecnológico. Las tecnologías de la información han experimentado un gran desarrollo gracias a los avances tanto en la tecnología de fabricación como en el diseño de sistemas arquitectónicos. Los cambios en la tecnología de fabricación han incrementado el nivel de integración de circuitos aumentando el número de componentes dentro de un chip, a la vez que se reduce el tamaño de los dispositivos. Ello ha favorecido el desarrollo y la evolución de los sistemas de propósito general y de los sistemas empotrados muy presentes en la vida cotidiana. Estos últimos no se circunscriben sólo al ámbito de los procesos industriales, hoy se encuentran en la mayoría de los dispositivos móviles disponibles en el mercado y han llegado a convertirse en un motor importante de crecimiento económico. La tendencia es que su presencia se extienda a todos los dispositivos electrónicos que nos rodean incrementando la computación ubicua. Sin embargo, la mayoría de los dispositivos móviles tienen como fuente de energía una batería, cuya capacidad y tamaño están limitados debido a las restricciones impuestas por su diseño.

La mayoría de los dispositivos actuales ha incrementado su funcionalidad dando soporte para aplicaciones multimedia, aplicaciones bien conocidas por su alto consumo de energía y la necesidad de alto rendimiento, lo cual incide negativamente en la vida de la batería y, en consecuencia, del dispositivo. De esta forma, uno de los principales objetivos que se plantea es incrementar el rendimiento al tiempo que se

reduce el consumo energético.

En definitiva, los sistemas actuales tienen como retos importantes, el consumo energético y la disipación del calor procedentes de los diferentes componentes que los forman. Con este objetivo, los científicos han propuesto y desarrollado algoritmos más eficientes y técnicas innovadoras de optimización tanto hardware como software que han permitido incrementar el rendimiento y reducir el consumo energético de los sistemas.

Uno de los componentes que más influye en el rendimiento y el consumo de energía es el subsistema de memoria [1, 2, 3]. Los investigadores han puesto el foco de atención sobre él como objetivo a optimizar y han sido numerosas las técnicas propuestas para ello. De esta forma, las diferentes propuestas de optimización han permitido mejorar el impacto sobre el rendimiento y el consumo de energía. Sin embargo, la ubicuidad se ha convertido actualmente en una propiedad imprescindible en las Tecnologías de la Información y de las Comunicaciones que ha propiciado grandes cambios tecnológicos en un periodo de tiempo muy corto. En este contexto, los problemas térmicos, la degradación del rendimiento y el elevado consumo de energía continúan siendo problemas importantes que afectan a los sistemas. En muchos casos, los diseñadores deciden que cuando se superan ciertos umbrales, los sistemas bajen la frecuencia de trabajo y, en consecuencia, el rendimiento. De esta forma, es posible reducir el impacto térmico y/o el consumo de energía, y así evitar daños irreversibles en los dispositivos. Además, un factor importante a tener en cuenta es el encarecimiento continuo de la energía y de los sistemas de refrigeración.

Desde esta perspectiva, resulta motivador estudiar el subsistema de memoria y, dado su nivel jerarquizado, abordar varios de los niveles que lo componen, proponiendo nuevas metodologías que permitan mejorar estos aspectos tan importantes. En este sentido, en esta tesis se aborda la optimización del impacto térmico del banco de registros del procesador, la optimización del rendimiento y el consumo energético del subsistema de memoria cache y por último, la gestión de memoria

dinámica con un nuevo método para diseñar gestores de memoria personalizados.

1.2. El subsistema de memoria

La jerarquía de memoria se ha identificado como un cuello de botella en cuanto al consumo de energía. Algunos autores sitúan su consumo en el 50 % de la energía consumida en el chip [4]. Los registros, que se encuentran situados más cerca del procesador, trabajan a gran velocidad pero su tamaño es pequeño y tienen un consumo de energía reducido, pero soportan un número considerable de accesos. Algunas arquitecturas aumentaron el número de registros y puertos, parámetros que determinan el tiempo de acceso y este se incrementa considerablemente. Además, esto provoca que aumente su tamaño, el tiempo de espera y el consumo de energía, lo cual afecta negativamente a la disipación de energía, la temperatura y el coste en los sistemas de refrigeración necesarios.

A continuación de los registros del procesador y en el siguiente nivel, se encuentra la memoria cache que es bien conocida por incrementar el rendimiento medio del procesador. Estudio previos sitúan su consumo de potencia entre el 20 % y 30 % de la potencia consumida en el chip [5]. El comportamiento de la memoria cache afecta tanto al rendimiento como al consumo de energía. Sin embargo, una configuración adecuada de cache permitiría mejorar el rendimiento y reducir el consumo de energía. Una configuración adecuada de cache se define por la selección de los parámetros óptimos de cache (tamaño de cache, tamaño de bloque, asociatividad, algoritmos de reemplazo, prebúsqueda, política de escritura, etc.), adaptados a cada aplicación a ejecutar. No obstante, la optimización para una aplicación individual no garantiza, en la mayoría de los casos, que otra aplicación mejore su rendimiento o consumo de energía.

Es importante resaltar que cada aplicación o tipo de aplicación presenta patrones de acceso a memoria diferentes. En consecuencia, se hace necesaria la exploración del espacio de búsqueda definido por las diferentes configuraciones. Cuando el espacio de búsqueda es muy amplio, esta tarea se convierte en inabordable mediante

un proceso de búsqueda exhaustiva. Esto lleva a las técnicas metaheurísticas a convertirse en firmes candidatas para realizar el proceso de exploración.

Por otra parte, tiempo de ejecución y consumo de energía son objetivos en conflicto. Así, un incremento de la asociatividad permite reducir los fallos y el tiempo de ejecución, pero produce un incremento de la complejidad hardware e incrementa el consumo de energía. Las propuestas presentadas por los investigadores se pueden englobar dentro de las técnicas de reconfiguración dinámica y de perfilado estático. La primera de ellas obtiene buenos resultados, pero incrementa la complejidad en el diseño del subsistema de memoria. Además, la mayoría de los trabajos han abordado un conjunto reducido de parámetros. La segunda se utiliza, generalmente, para realizar el análisis del comportamiento y realizar, posteriormente, propuestas de mejoras basadas en los resultados obtenidos. El perfilado estático se utiliza en esta tesis para recoger los patrones de acceso a memoria de cada aplicación, datos necesarios para abordar el problema de optimización.

El siguiente nivel a abordar en la jerarquía de memoria es la memoria principal y, particularmente, la asignación de memoria a los programas en tiempo de ejecución que se lleva a cabo por parte de los gestores de memoria dinámica. La selección del gestor de memoria dinámica no es una tarea trivial. Resolver una petición de memoria dinámica es una tarea compleja y el algoritmo de asignación debe mitigar el problema de la fragmentación tanto interna como externa. Así, medir de forma adecuada el coste y la eficiencia de un gestor de memoria dinámica es fundamental para incrementar el rendimiento del sistema.

Además, las características de las aplicaciones multimedia que se ejecutan en los dispositivos actuales (de propósito general y empotrados), requieren gran cantidad de asignación de memoria dinámica. De esta forma, implementar el gestor de memoria dinámica de acuerdo al conjunto de aplicaciones o a un tipo de aplicación particular para un sistema es fundamental para incrementar su rendimiento. Es posible seleccionar un gestor de memoria dinámica que sea muy rápido en la asignación de memoria, pero que trabaje de forma ineficiente y, por tanto, afecte negativamen-

te al rendimiento del sistema. En esta tesis se aborda el análisis de los gestores de memoria dinámica y se propone una metodología de optimización multi-objetivo basada en programación genética para diseñar gestores de memoria dinámica que se adapten a las características de cada aplicación.

1.3. Objetivos y contribuciones

En esta tesis se considera el problema de la optimización del subsistema de memoria que es posible aplicar tanto a sistemas de propósito general como a sistemas empotrados, presentes en la mayoría de los dispositivos móviles actuales. A continuación se resumen las principales aportaciones que esta tesis realiza.

Gestión del Banco de Registros La contribución de la tesis en este punto se divide en dos partes, la primera de las cuales ha sido la realización de un análisis exhaustivo del incremento de la temperatura a consecuencia de los accesos a la estructura interna del banco de registros y, de esta forma, comprobar cuál es su influencia sobre la temperatura del microprocesador. La segunda ha consistido en la aplicación de un algoritmo evolutivo multi-objetivo que permite distribuir los accesos a lo largo del banco de registros, de tal manera que aquellos registros con un mayor número de accesos se encuentren alejados entre sí y, por tanto, permita reducir el incremento de la temperatura debido a la transferencia de calor.

Memoria Cache Otro de los objetivos de esta tesis es la optimización del subsistema de memoria cache, el cual se ha abordado proponiendo un marco de optimización estructurado en tres fases: (1) caracterización de la memoria cache; (2) obtención de las trazas de las aplicaciones objetivo; (3) ejecución del algoritmo de optimización que utiliza tres técnicas de computación evolutiva diferentes: Gramáticas Evolutivas (GE, *Grammatical Evolution*), Algoritmo Multi-objetivo NSGA-II y Evolución Diferencial (DE, *Differential Evolution*). Así, el marco de optimización permite buscar automáticamente la mejor configuración de cache para un conjunto

de aplicaciones, de forma individual o conjunta, en un sistema empotrado y/o de propósito general. Las aportaciones que esta tesis realiza son:

- Proceso de optimización dirigido por gramáticas evolutivas y perfilado estático, que permite encontrar la mejor configuración de cache para un conjunto de aplicaciones objetivo. Está dividida en tres fases, la primera de las cuales se encarga de la caracterización hardware de las memorias cache, la segunda obtiene trazas de las aplicaciones y, la tercera ejecuta el algoritmo GE y obtiene un conjunto de buenas configuraciones para cada aplicación individual. La bondad de cada configuración se calcula ponderando el tiempo de ejecución y el consumo de energía frente a una configuración de referencia. Promediando el porcentaje de mejora de todas las soluciones compartidas por las aplicaciones objetivo, los resultados alcanzan una mejora en promedio del 62 % con respecto a una configuración cache de la arquitectura ARM9, tomada como referencia. Además, memorizar los individuos ya evaluados permite reducir el tiempo de evaluación alrededor de un 93, 6 %, lo cual supone realizar el proceso de optimización en un tiempo razonable.

- Proceso de optimización dirigido por un algoritmo multi-objetivo y perfilado estático que tiene como objetivo encontrar el mejor conjunto de configuraciones de cache para un conjunto de aplicaciones objetivo. Este conjunto de configuraciones es aquel que reduce el tiempo de ejecución y el consumo de energía para cada aplicación, y por tanto, mejora el rendimiento e incrementa el tiempo de vida tanto de las baterías como de los dispositivos. Además, este enfoque facilita la tarea de los diseñadores de sistemas cuando tienen que seleccionar la memoria cache a implementar en un sistema objetivo. Para cada aplicación individual se dispone de un conjunto de configuraciones de cache ya identificadas como configuraciones que ofrecen buen rendimiento.

  Tomando una configuración base de referencia, típica de la arquitectura ARM9, los resultados experimentales muestran un porcentaje de mejora promedio del

64, 43 % y 91, 69 % en el tiempo de ejecución y consumo de energía, respectivamente. Con la finalidad de validar los resultados obtenidos, se toman como referencia dos nuevas configuraciones cache utilizadas en las arquitecturas cortex-A9 y cortex-A15. En este caso, se obtiene una mejora en tiempo de ejecución del 24, 15 % y 16, 85 %, y en consumo de energía del 88, 90 % y 89, 8 %, respectivamente. A continuación se establece el mismo espacio de búsqueda y configuración de referencia utilizados en GE y los resultados presentan un conjunto de configuraciones de cache que mejoran el tiempo de ejecución en un 33, 87 % y el consumo de energía en un 71, 79 %.

- Proceso de optimización dirigido por DE y basado también en perfilado estático, cuya principal contribución es la búsqueda de la mejor configuración cache que se adapte a las necesidades de un conjunto completo de aplicaciones objetivo. La mejor configuración se define como aquella que alcanza el mejor valor de la función objetivo, de la que forman parte el tiempo de ejecución y consumo de energía, para todas las aplicaciones. Los resultados obtenidos aportan un porcentaje de mejora cercano al 65 % respecto a la configuración base basada en las especificaciones de la arquitectura ARM9.

El marco de optimización presentado contribuye a mejorar la toma de decisiones por parte de los diseñadores de sistemas. Una vez conocido el tipo de aplicación o aplicaciones a ejecutar en un dispositivo, es posible analizar el comportamiento de estas y en función de los resultados seleccionar la configuración cache que mejor se adapte a ellas. Además, el proceso sólo se ha de ejecutar una vez por cada conjunto de aplicaciones objetivo.

Gestores de Memoria Dinámica En esta tesis, se propone un nuevo método de optimización dirigido por un algoritmo GE que contribuye a facilitar la evaluación de DMM multicapa complejos y permite diseñar DMM adaptados a las características de aplicaciones reales intensivas de memoria. Dado que cada aplicación presenta necesidades de memoria dinámica diferentes, la definición de la gramática se realiza

de forma personalizada a partir de la traza de cada aplicación. La evaluación de cada DMM se realiza también en base a la traza de la aplicación, sin necesidad de integrar el DMM en la aplicación objetivo. En esta metodología cada DMM diseñado se compara con cinco DMM de propósito general para un conjunto de seis aplicaciones altamente dinámicas. Los resultados presentan una ahorro promedio en tiempo de ejecución y consumo de energía del $59,27\,\%$ y $30,62\,\%$, respectivamente.

Así, el análisis realizado en esta tesis ofrece a los diseñadores de sistemas operativos un entorno para seleccionar el mejor gestor de memoria personalizado y optimizado según las necesidades de cada aplicación, después de ser evaluado con cinco DMM de propósito general y las métricas de rendimiento y uso de memoria, generalmente utilizadas por los diseñadores de DMM.

1.4. Organización

Esta tesis estudia la optimización del rendimiento y el consumo energético de un componente esencial en el diseño de microprocesadores, como es el subsistema de memoria. El banco de registros, el subsistema de memoria cache y la memoria principal, a través de la gestión de memoria dinámica, son tres de los elementos más significativos dentro del subsistema de memoria. El diseño adecuado de todos y cada uno de ellos tiene un gran impacto tanto en el rendimiento como en el consumo de energía, y por tanto, será objeto de optimización.

Atendiendo a la particularidad del enfoque propuesto para cada problema, se han utilizado técnicas de computación evolutiva para abordar el proceso de optimización, debido a los amplios espacios de búsqueda utilizados.

Para exponer el trabajo realizado, esta tesis se organiza como sigue: en el Capítulo 2 se presentan las principales características del subsistema de memoria. Se realiza un recorrido histórico y se presentan las arquitecturas utilizadas en esta tesis, así como las innovaciones tecnológicas en el campo de los microprocesadores y que han propiciado nuevos retos para la optimización.

El Capítulo 3 se centra en revisar las principales técnicas metaheurísticas utili-

zadas para abordar los problemas de optimización y, particularmente, las técnicas de computación evolutiva, ampliamente utilizadas en esta tesis.

El banco de registros, como uno de los componentes que más energía consume, es objeto de estudio en el Capítulo 4. Concretamente, el estudio se centra en la gestión térmica del banco de registros y la influencia que tiene en la temperatura del procesador. Así, temperatura y consumo energético se definen como objetivos a optimizar, lo que hace necesario la utilización de un algoritmo multi-objetivo.

El Capítulo 5 tiene como objetivo la optimización del subsistema de memoria cache con un único nivel de cache. La correcta configuración de la memoria cache, definida por la selección adecuada de los parámetros que la definen, afecta al rendimiento y al consumo energético. Para el primer enfoque propuesto en la sección 5.4, la definición de una única función objetivo hace que no sea necesario la aplicación de un MOEA. Bajo estas premisas, GE se presenta como una buena alternativa que ha demostrado su capacidad para representar configuraciones de cache correctas, donde el fenotipo se forma con los parámetros de configuración. Por otra parte, GE añade flexibilidad al adaptar la expresión de un individuo (fenotipo) al formato de la llamada a un simulador de cache, que se integra en el proceso de optimización. Así, si se desea cambiar el simulador sólo sería necesario cambiar la gramática.

Sin embargo, en el problema se identifican claramente dos objetivos, por tanto en la sección 5.5 se aborda el problema utilizando un algoritmo multi-objetivo que a través del espacio de búsqueda encuentre aquellas configuraciones que mejoren el rendimiento (reduciendo el tiempo de ejecución) y reduzcan el consumo de energía. Ambas medidas son relativas a las operaciones que se realizan sobre la memoria cache. De hecho, se define el consumo energético y el tiempo de ejecución debido a las operaciones de memoria cache, como objetivos a optimizar. El indicador hipervolumen se utiliza para comparar la calidad de las soluciones obtenidas.

Las dos secciones previas estudian la optimización de la memoria cache para cada aplicación individual. En la sección 5.6 se aborda este proceso de optimización buscando la mejor configuración de cache, aquella que reduzca el tiempo de ejecu-

ción y el consumo de energía, para el conjunto de aplicaciones objetivo completo de forma simultánea. Dado el coste computacional requerido, se ha seleccionado Evolución Diferencial entre cuyas características se encuentra la rapidez y sencillez, y se han seleccionado seis aplicaciones para su estudio.

Para finalizar con la optimización de la memoria cache llevada a cabo en esta tesis, en la sección 5.7 se realiza la comparación entre los resultados obtenidos mediante GE y NSGA-II. Para lo cual, antes se procede a realizar las pruebas mediante NSGA-II, en las mismas condiciones que GE. De esta forma, es posible comparar no sólo con GE, sino también entre los dos resultados obtenidos con NSGA-II.

El Capítulo 6 aborda la memoria principal, de forma particular la gestión de memoria dinámica. Las aplicaciones multimedia, que actualmente se ejecutan sobre muchos de los dispositivos móviles existentes, realizan una gran cantidad de operaciones dinámicas debido al tipo de datos con el que trabajan. De hecho, el Gestor de Memoria Dinámica tiene una influencia considerable en el comportamiento de la aplicación y del hardware. En esta tesis se presenta una metodología multi-objetivo basada en GE para proponer un DMM a medida de las necesidades de cada aplicación. Se simplifica el proceso de evaluación del comportamiento de los gestores de memoria de propósito general y/o personalizados, utilizando aplicaciones con necesidades exigentes de memoria dinámica.

Por último, en el Capítulo 7 se presentan las conclusiones alcanzadas tras la realización de este trabajo de investigación y las posibles líneas de trabajo futuro.

Capítulo 2

Subsistema de memoria

El diseño de computadores ha evolucionado a gran velocidad desde la aparición del primer ordenador. A pesar de los cambios continuos que la ley de Moore predijo, el principal objetivo es alcanzar el rendimiento óptimo del procesador. Las mejoras de las tecnologías aplicadas a los procesos de fabricación, los avances en el diseño arquitectónico con la implementación de innovadoras técnicas hardware y software o el desarrollo de algoritmos más eficientes, entre otros, han contribuido a incrementar el rendimiento, la fiabilidad y la velocidad de los computadores, además de mejorar la seguridad y reducir el consumo energético.

Uno de los componentes, cuya influencia sobre el rendimiento y el consumo de energía ha quedado suficientemente demostrada, es el subsistema de memoria [6, 7, 1, 2, 3, 8, 9]. Este capítulo se dedica a mostrar una perspectiva histórica de los principales avances en el campo de los microprocesadores, que han permitido alcanzar el estado actual. Particularmente, nos centramos en la evolución del subsistema de memoria y algunas de las técnicas de optimización aportadas por la comunidad científica durante las últimas décadas, para abordar a continuación la optimización del subsistema de memoria.

2.1. Perspectiva histórica

La innovación tecnológica y la necesidad de mantener la compatibilidad con diseños anteriores ha dirigido el desarrollo de arquitectura de computadores. El incremento de la velocidad de reloj, la aparición de la *segmentación* de instrucciones

(desarrollada por IBM para la arquitectura RISC (*Reduced Instructions Set Compiler*), a mediados de los 80), y la incorporación de las instrucciones vectoriales, propiciaron el incremento del paralelismo *on-chip*, permitiendo explotar el paralelismo a nivel de instrucción (ILP, *Instruction Level Paralelism*) y el paralelismo a nivel de tarea (TLP, *Thread Level Parallelism*) [10].

Los procesadores superescalares y los procesadores vectoriales son dos ramas de la evolución en el diseño de arquitectura de computadores. Los procesadores superescalares, que permiten minimizar los riesgos estructurales que se dan en procesadores segmentados, explotan el paralelismo a nivel de instrucción al disponer de varias unidades funcionales e implementar técnicas hardware para la ejecución de instrucciones en paralelo. Sin embargo, deben aportar soluciones a las dependencias de datos entre instrucciones y garantizar que las instrucciones se ejecuten en el orden correcto.

Los procesadores vectoriales se diseñaron para dar soporte a aplicaciones del campo científico e ingenieril que manejan grandes volúmenes de datos almacenados en matrices y vectores de gran tamaño, donde el tratamiento de los datos en paralelo incrementa considerablemente el rendimiento.

La aparición de la arquitectura VLIW (*Very Long Instruction Word*) reduce de forma significativa el número de ciclos por instrucción (CPI, *Cycles Per Instruction*). VLIW dispone de un tamaño de instrucción muy grande. En este caso, el compilador es el encargado de planificar la ejecución en paralelo y minimizar los retardos debidos a las dependencias de datos entre instrucciones, a diferencia de los procesadores superescalares.

Junto a los enfoques anteriormente mencionados, a lo largo de los años se han ido sumando otras técnicas de optimización como la ejecución fuera de orden, planificación dinámica, ejecución especulativa y predicción de salto, entre otras, que han tratado de minimizar el efecto negativo de las dependencias de control y de datos en el rendimiento del procesador.

Sin embargo, el bajo ILP implícito en una tarea, por la propia programación de

los algoritmos, limita la mejora del rendimiento en los procesadores superescalares. Por otra parte, las arquitecturas multitarea permiten que los recursos no utilizados por la tarea actual, puedan ser asignados a otra y así, ocultar las latencias debidas a determinadas operaciones. Enfoques como multitarea de grano fino (FMT, *Fine-grained Multithreading*), multitarea de grano grueso (CMT, *Coarse-grained Multithreading*), multitarea simultanea (SMT, *Simultaneous Multithreading*) o multiproceso a nivel de chip (CMP, *Chip Multiprocessor*) soportan multitarea on-chip.

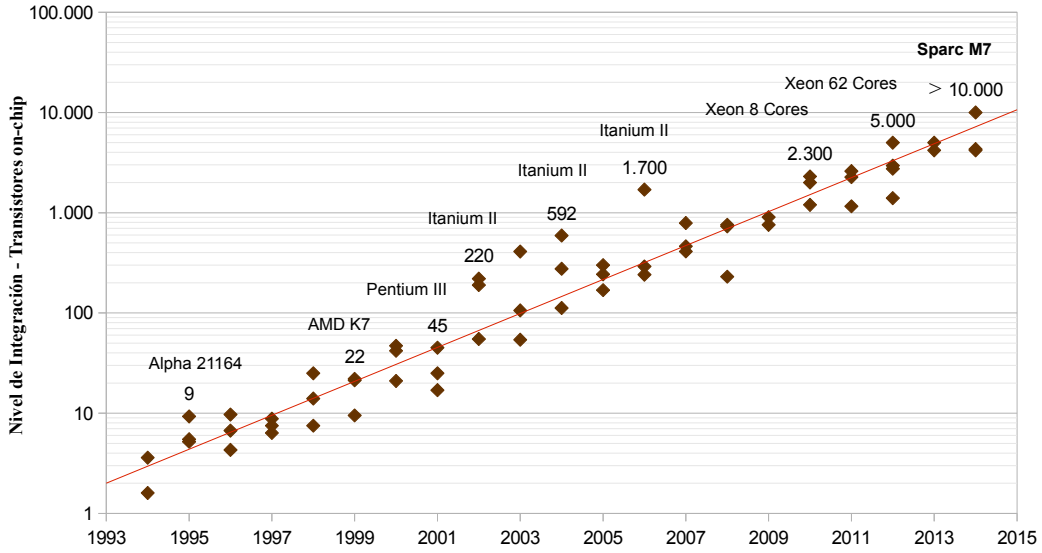


Figura 2.1: Evolución en las dos últimas décadas de la integración en chip. Los datos se representan en millones de transistores. La Ley de Moore predijo que el número de componentes en un circuito integrado se duplicaría aproximadamente cada dos años, mientras el tamaño del chip se reduciría. En 2012 el procesador Intel Core-i7 alcanzaba los 2200 millones de transistores, con una tecnología de fabricación de 22nm. En 2014, el procesador Sparc M7 de Oracle superó los 10000 millones de transistores.

La técnica de SMT [11, 12] permite que varias tareas envíen instrucciones a las unidades funcionales, en un mismo ciclo de reloj. El número de ciclos que el procesador permanece inactivo se reduce a la vez que se aprovecha el TLP, cuando no es posible aprovechar el ILP.

Ha sido en las dos últimas décadas [13], cuando las mejoras tecnológicas han llevado al incremento del número de transistores por chip, tal y como se muestra en la Figura 2.1, lo que ha permitido situar todos los niveles de la jerarquía de memoria

cache dentro del chip del procesador, incrementar la capacidad de almacenamiento, aumentar la frecuencia de reloj como se observa en la Figura 2.2, y proporcionar ciclos de reloj más rápidos. No obstante, mayor densidad de integración supone un incremento en el consumo de potencia.

Además, una mayor frecuencia de trabajo también tiene un gran impacto sobre el consumo de energía y por tanto en la disipación de potencia, factores ambos que afectan a la fiabilidad de los materiales y por consiguiente al tiempo de vida de los dispositivos [14, 15]. La necesidad de preservar los dispositivos lleva a utilizar una frecuencia menor y por consiguiente, a reducir el voltaje para controlar el consumo de potencia. El precio a pagar es una pérdida de rendimiento que, sin embargo, se minimiza gracias al incremento de paralelismo que proporcionan los sistemas de múltiples núcleos o *multicores*.

Por ejemplo, la familia de procesadores Intel Core i7 en su cuarta generación funciona con frecuencia de reloj de 4 GHz, aunque puede llegar a una frecuencia máxima de 4.4GHz. Además, disponen de 4/8 núcleos por procesador, es decir 4 cores o unidades de procesamiento independientes en un solo chip, y 8 hebras, donde una hebra se define como la unidad mínima (conjunto de instrucciones ordenado), que puede ser asignada y procesada por un núcleo de la CPU. El incremento del número de núcleos dentro del mismo chip está aportando mejoras sustanciales a los sistemas de cómputo. La Figura 2.3 presenta la evolución del número de núcleos a lo largo de las dos últimas décadas incrementando la capacidad de procesamiento. Aunque cada núcleo tiene recursos propios (registros, unidades de ejecución, uno o varios niveles de cache, etc.), otros recursos son compartidos por todos los núcleos, como sucede con la memoria principal y en muchas casos con el segundo nivel de cache (L2), por ejemplo en el Intel® $Core^{TM}2$ Duo. De esta forma, la contención de estos recursos compartidos tiene un impacto significativo sobre el rendimiento y el consumo de energía [16].

El subsistema de memoria, jerárquicamente estructurado, es uno de los componentes que más rédito ha sacado de los avances tecnológicos durante los últimos

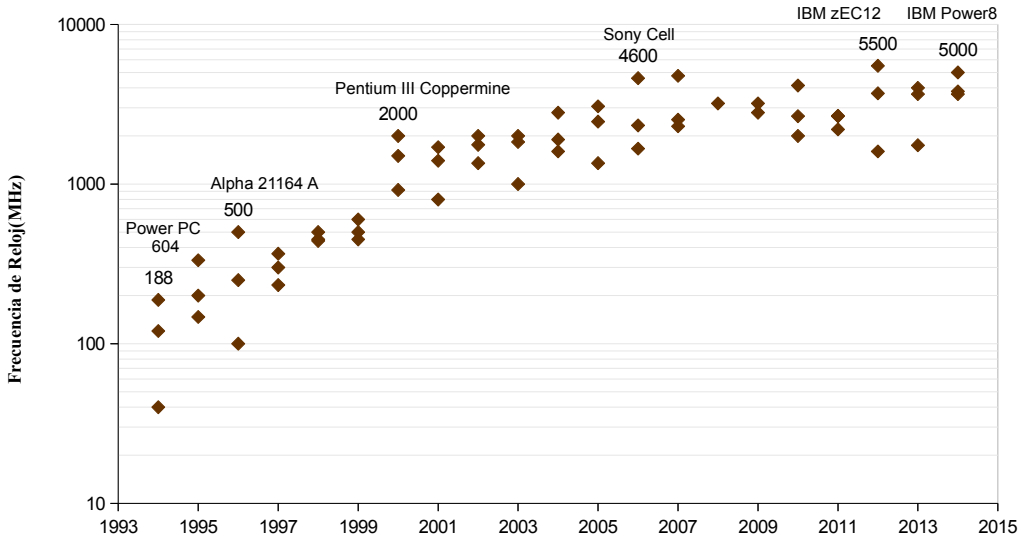


Figura 2.2: Evolución de la frecuencia de reloj durante las dos últimas décadas. La frecuencia, reflejada en el eje y, a escala logarítmica se representa en MHz y hacen referencia a las frecuencias de reloj disponibles en el periodo indicado. Durante la década de los noventa se observa cómo la frecuencia de reloj se multiplicó por 20. El procesador IBM z Systems alcanzaba los 5.5 GHz [17].

años. Aquellos elementos del subsistema de memoria situados junto al procesador trabajan a su misma velocidad con un consumo de energía reducido, pero son de pequeño tamaño. Los registros se encuentran en el nivel más alto de la jerarquía de memoria, dentro del chip del procesador. Su tamaño es menor que el de la memoria cache y soportan accesos constantes con un consumo de energía menor [10].

La aparición de la arquitectura RISC [10] y sus formatos de instrucción permiten múltiples operaciones por instrucción. La presentación de un tamaño de instrucción fijo y un número reducido de formatos de instrucción posibilita la segmentación y el paralelismo en la ejecución de instrucciones. El objetivo es reducir los accesos a memoria que suponen una alta penalización en cuanto a tiempo de acceso y aumentar el uso de registros cuyo acceso no tiene penalización dada la cercanía a la CPU. Como consecuencia se incrementa la presión ejercida sobre los bancos de registros. Por una parte, la arquitectura RISC necesita que los datos estén en el banco de registros para su procesamiento e incrementa el número de operaciones a realizar en paralelo. Una operación de lectura de RAM permite rellenar varios registros y una vez realizada la operación, los operandos y el resultado se conservan

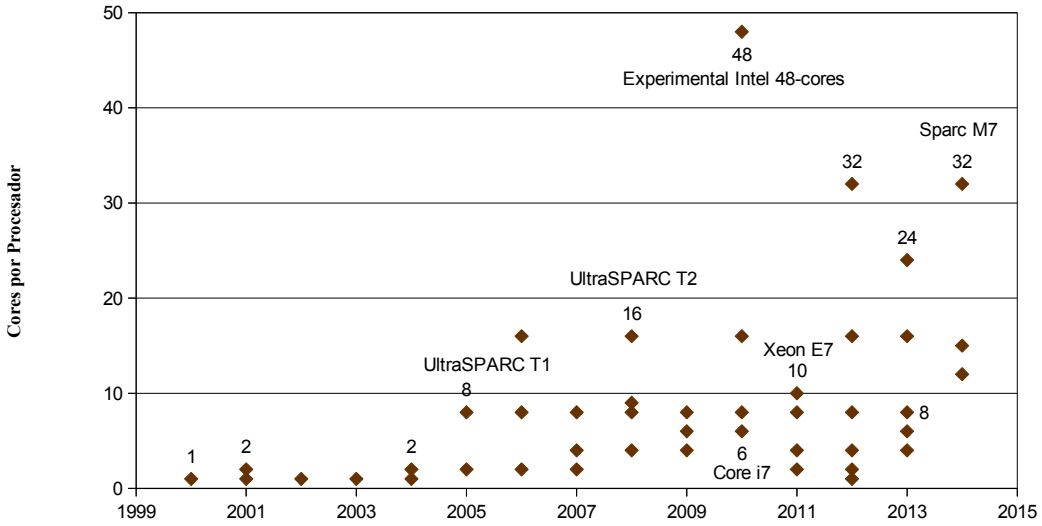


Figura 2.3: Evolución del número de núcleos por procesador durante la última década. La tecnología del silicio está próxima a alcanzar el límite de rendimiento, y se hace necesario aportar soluciones al aumento de la capacidad de procesamiento. La replicación del núcleo de procesador mejora el rendimiento en los sistemas basados en microprocesador system-on-chip (SoC's). Una tecnología que se presenta prometedora es 3D-IC packaging gracias a la evolución de las tecnologías de integración y de empaquetado.

en memoria, facilitando la ejecución de otras operaciones. Para dar soporte a los nuevos requisitos es necesario explotar el paralelismo, lo que lleva al aumento del número de registros y puertos disponibles. Dado que número de registros y puertos determinan el tiempo de acceso, este sufre un importante incremento. Como resultado se incrementa el tamaño, el tiempo de espera y el consumo de energía, lo cual, junto a la mayor integración de circuitos, incide considerablemente en la disipación de energía, la temperatura del microprocesador y el coste debido a los sistemas de refrigeración.

Una parte de esta tesis se dedica a analizar la influencia de los accesos al banco de registros sobre la temperatura total del microprocesador. Otros autores han demostrado cómo en función de una optimización es posible realizar diseños que incrementen el rendimiento del procesador y/o reduzcan el consumo de energía y el incremento de temperatura.

En [18] se propone la división de los registros de procesador en varios bancos de

registros, cada banco puede contener un número diferente de registros, puertos y por tanto diferentes tiempos de acceso. Se presentó una propuesta con múltiples bancos de registros en un sólo nivel, así como un diseño con 2 niveles, sobre arquitectura RISC. Cada registro lógico se mapea a uno de los registros físicos pertenecientes a uno de los bancos. Los bancos de registros permiten asociar una misma dirección a varias copias de un registro. En la Figura 2.4 se representa cómo la familia de procesadores ARM asocia los diferentes modos de trabajo del procesador a diferentes bancos de registros [19]. El enfoque está desarrollado especialmente para arquitecturas RISC. La asociación entre bancos de registros y modos del procesador acelera los cambios de contexto e incrementa el rendimiento del microprocesador [20].

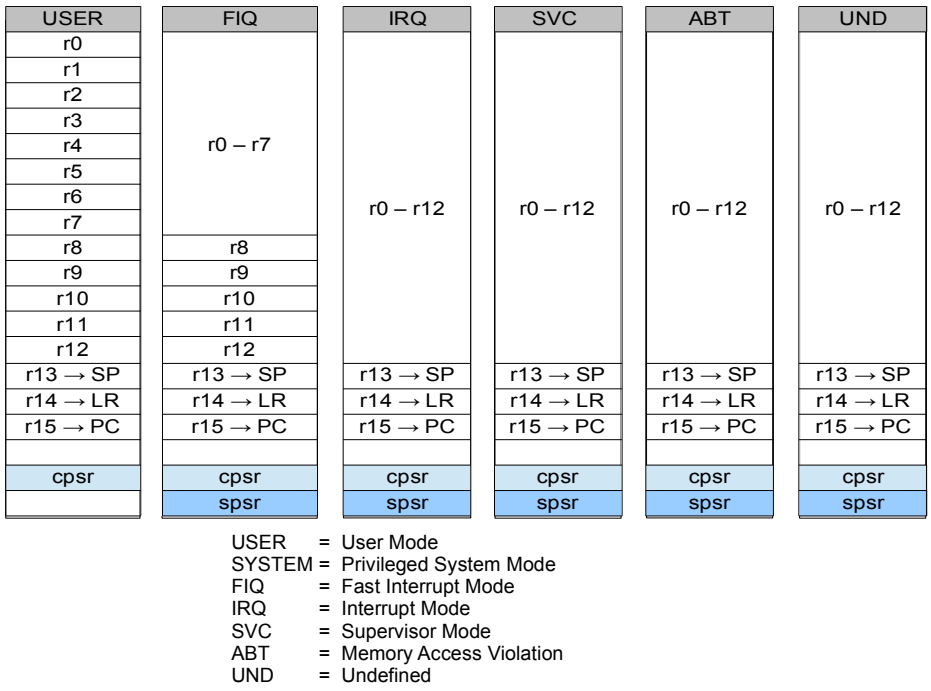


Figura 2.4: Banco de registros de un procesador de la arquitectura ARM9, basada en RISC y según los 7 modos del procesador [19]. El sistema utiliza los mismos registros que el modo usuario, pero en modo privilegiado.

El procesamiento de instrucciones explícitamente en paralelo (EPIC, *Explicitly Parallel Instruction Computing*), o la generación de código eficiente proporcionadas por el compilador tratan de minimizar el impacto sobre el rendimiento, consumo de energía y temperatura, tanto para arquitecturas RISC como VLIW.

Además de los registros, el sistema de memoria está compuesto por la memoria

cache (jerárquicamente dividida en uno o varios niveles), la memoria principal y el almacenamiento secundario. El principal objetivo de los científicos con respecto al sistema de memoria es realizar un diseño de memoria muy rápida, con gran capacidad y cuyo coste sea bajo.

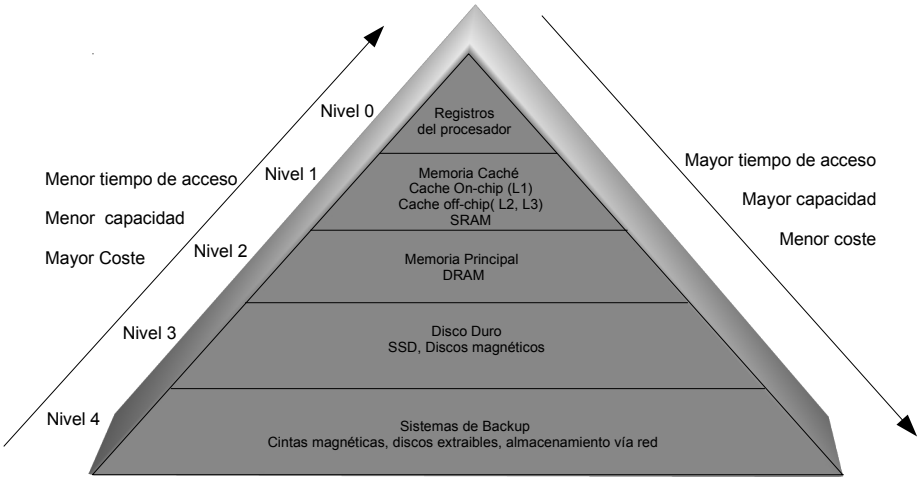


Figura 2.5: Diagrama general de la jerarquía de memoria de un sistema computador. El nivel 0 indica mayor cercanía al procesador y por tanto, un menor tiempo de acceso, pero también menor capacidad a mayor coste. Conforme se incrementa el nivel, la capacidad aumenta y se reduce el coste, pero crece el tiempo de acceso.

El subsistema de memoria se organiza jerárquicamente por tamaño y velocidad [10]. Los componentes de memoria más cercanos al procesador son más rápidos y pequeños y su coste es mayor. Cuanto más se aleja un componente de memoria del procesador tiene mayor capacidad y es más económico, aunque es más lento. Sin embargo, los accesos a memoria se rigen por el principio de localidad que utiliza este esquema jerarquizado, como se observa en la Figura 2.5, para mejorar el rendimiento. El principio de localidad explica que la distribución de los accesos a memoria no es uniforme a través del espacio de direcciones completo, sino que en un momento determinado los accesos se agrupan alrededor de un área específica. El principio de localidad fue definido formalmente en [21] y ha sido ampliamente sustentado [22]. Se presenta de dos formas: temporal y espacial.

- La localidad temporal incide en el carácter repetitivo de los programas, don-

de una instrucción ejecutada o un dato accedido recientemente tiene mayor probabilidad de volver a ser ejecutada o accedido en un futuro próximo.

- La localidad espacial hace referencia al hecho de que el acceso a una instrucción o dato, incrementa la probabilidad de acceso de aquellas instrucciones o datos más cercanos, en un breve espacio de tiempo.

Según el principio de localidad, una pequeña parte de los datos situados en la memoria principal se llevan a la memoria cache. Como resultado, las direcciones de un nivel de memoria de mayor capacidad, pero más lento, se mapean al siguiente nivel de memoria de menor capacidad pero más rápido, mejorando el rendimiento medio del sistema. El proceso de obtención de un dato comienza con la búsqueda, por parte de la CPU, del dato en la memoria cache. Si se encuentra en la memoria cache, el tiempo de acceso es muy pequeño. En caso contrario, se produce un fallo de cache y se inicia la búsqueda del dato en memoria principal, el resultado es la penalización en cuanto al tiempo de acceso por la lejanía de la memoria principal del procesador.

La memoria cache utiliza tradicionalmente la tecnología SRAM (*Static Random Access Memory*). La memoria principal se implementa con tecnología DRAM (*Dynamic Random Access Memory*). Para almacenamiento secundario se utilizan discos magnéticos y SSD (*Solid State Disk*), basados en tecnología flash-NAND proporcionando un gran rendimiento y cuyo coste se ha reducido en los últimos años. De hecho, las soluciones híbridas han ganado peso, como se desprende de estudios realizados [23, 24]. Por otra parte, varias tecnologías han aparecido en el mercado con el objetivo de conseguir el ideal de una memoria de gran capacidad, rápida y con un menor consumo de energía. STTRAM (*Spin Torque Transfer Random Access Memory*) [25], PCRAM (*Phase Change Random Access Memory*) [26], RRAM (*Resistive Random Access Memory*) [27] o NEMM (*Nanoelectromechanical Memory*) [28], se presentan como alternativas fiables en la jerarquía de memoria, caracterizadas principalmente por ser no volátiles. Aunque no son objeto de estudio en esta tesis, su mención es fundamental dada la relevancia que están teniendo

actualmente y la que, con toda seguridad, van a tener en el futuro inmediato.

Los requisitos en el diseño de microprocesadores, aunque se utilice la misma tecnología, presentan importantes diferencias según el sector al cual vaya dirigido: sistemas de cómputo personal, servidores, supercomputadores o sistemas empotrados. Mientras que un sistema de cómputo personal es generalmente utilizado por un único usuario, los servidores deben soportar múltiples usuarios y ejecución concurrente de aplicaciones muy diversas. Los supercomputadores suministran los recursos necesarios para cómputo de alto rendimiento y necesidades de memoria [10]. De hecho, las diferencias más importantes las encontramos con respecto a los sistemas empotrados, los cuales son el objetivo de estudio en este trabajo de tesis y de los que se hablará en la sección 2.1.1.

La memoria principal, situada en el nivel más bajo de la jerarquía de memoria, se encarga de responder a las peticiones que realiza la memoria cache. Implementada habitualmente con tecnología DRAM, la latencia y el ancho de banda tienen un gran impacto a consecuencia de los fallos en la cache. Otros factores inciden también en el rendimiento del sistema como son la política de reemplazo [1], algoritmo de búsqueda [2] y la política de escritura [3]. Todos ellos han sido ampliamente estudiados en las últimas décadas [29, 30, 31].

Además, la incorporación de varios niveles de cache ha permitido reducir la latencia y mejorar el ancho de banda. Otras técnicas, como la organización de los chips de memoria en bancos, que facilitan la realización de varias operaciones en paralelo o la utilización de controladores de memoria diferentes para cada banco también mejoran el ancho de banda.

Por otra parte, el sistema de memoria virtual apareció ante la necesidad de aportar protección en sistemas donde varios procesos se ejecutan de forma concurrente. Así, los niveles más bajos de la jerarquía de memoria, memoria principal y almacenamiento secundario, son gestionados automáticamente. La memoria virtual elimi-

[1] Algoritmo que determina el bloque a desalojar de la cache cuando no hay espacio.

[2] Algoritmo que decide cuándo y qué bloque se lleva de la memoria principal a la memoria cache.

[3] Algoritmo para decidir cuándo se actualiza la memoria principal con los datos modificados de la cache.

na las restricciones de tamaño de programa debido al tamaño de la memoria principal y divide la memoria física en bloques que son asignados a los procesos, como se refleja en la Figura 2.6. Las técnicas de paginación, segmentación, políticas de reemplazo, etc. son componentes cruciales de este sistema de memoria virtual, que incrementan el rendimiento explotando el principio de localidad referencial (temporal y/o espacial), de los programas.

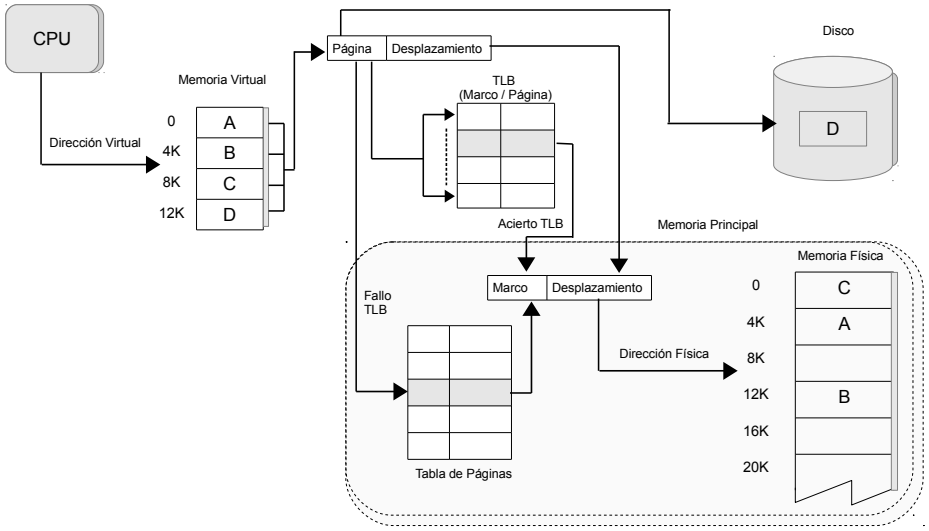


Figura 2.6: Ejemplo de mapeo entre la memoria física y la memoria virtual. Basado en el esquema presentado en [10], donde la memoria se divide en bloques de 4K y se muestra la ubicación de 4 bloques, de los cuales 3 de ellos se encuentran cargados en memoria principal (A, B y C) y otro en disco (D). El proceso de conversión de una dirección lógica a una física se realiza mediante una combinación de hardware y software. La página se busca inicialmente en el Bufer de Traducción Adelantado o TLB (*Translation Lookaside Buffer*), si no se encuentra se busca en la Tabla de Páginas. En ambos casos, se retorna del marco de memoria física donde se encuentra la página buscada, cuya posición dentro del marco se completa con el desplazamiento. Si los dos pasos anteriores no tuvieran éxito, hay que buscar la página en el almacenamiento secundario.

A finales de la década de los cincuenta (cuando se comienza a desarrollar el lenguaje de programación LISP y en 1960 la técnica denominada *Recolector de Basura*), se introduce la asignación de memoria dinámica aportando una flexibilidad que la asignación estática no proporcionaba. En una asignación estática de memoria el tamaño de las estructuras de datos de un programa se debe conocer a priori. La asignación o liberación dinámica de memoria se realiza desde/hacia el área de memoria dinámica, también denominada montículo, y se produce durante la ejecución de un proceso. En la actualidad, todos los sistemas operativos proporcionan asig-

nación dinámica de memoria [32] y son los gestores de memoria dinámica (DMM, *Dynamic Memory Management*), los encargados de dicha tarea.

Un DMM responde a las peticiones de memoria que realizan las aplicaciones y asigna el bloque libre que mejor se ajuste (según la política de asignación) al tamaño solicitado y que minimice el problema relativo al desaprovechamiento de la memoria. Según el algoritmo implementado y el uso que las aplicaciones hacen de la memoria dinámica, el DMM puede causar un gran impacto en el rendimiento medio del sistema. Además, las aplicaciones multimedia tienen unas necesidades muy altas en cuanto a la utilización de recursos y con la expansión de los dispositivos móviles, la presión sobre el subsistema de memoria dinámica se ha ido incrementando y, por consiguiente, la necesidad de aportar optimizaciones. En la sección 2.2 se referencian algunos de los estudios en el campo de la gestión de memoria dinámica y en el Capítulo 6 se propone una nueva metodología para el diseño de gestores de memoria dinámica a medida de una aplicación objetivo, la cual analiza el impacto del DMM sobre la aplicación utilizando las métricas de rendimiento y uso de memoria.

2.1.1. Evolución de los sistemas empotrados

Los sistemas empotrados están presentes en nuestra vida diaria. Telefonía móvil, automoción, control industrial, aparatos electrónicos, sistemas militares, aeroespaciales, médicos, etc., han cambiado nuestra vida en muchos aspectos [33]. Cualquier dispositivo electrónico puede contener un sistema empotrado que se encarga de realizar alguna función específica. A diferencia de los sistemas de propósito general, los sistemas empotrados están especialmente diseñados para realizar un reducido número de funciones o una única función. La complejidad dependerá del tipo de sistema para el que se diseñe y de la tarea a realizar. El diseño se realiza bajo estrictos requisitos de tamaño, coste y consumo de potencia y energía, debido al limitado número de recursos disponibles como son los componentes de la CPU, memoria (RAM), tamaño de la memoria ROM o Flash, velocidad del procesador y periféri-

cos [34].

Dentro de los sistemas empotrados encontramos importantes diferencias, según la función para la que se diseñen, por ejemplo, si el sistema debe ejecutar en tiempo real o no debe hacerlo, presencia de microprocesadores, sistema operativo, etc. Aquellos especialmente dirigidos a su uso dentro de un Sistema en Tiempo Real (RTS, *Real Time System*), tienen que cumplir con severas restricciones en cuanto al tiempo de respuesta y en muchos casos, esta depende de la información del entorno obtenida mediante sensores (temperatura, humedad, posición, decisiones del usuario, etc.) y actuadores que tras analizar dicha información realizan las acciones adecuadas. Otras veces, son sistemas críticos que controlan, por ejemplo, transporte de viajeros, actuadores médicos, etc., y cuyo comportamiento debe ser determinista y bajo estrictas condiciones de seguridad y fiabilidad. Un fallo en el sistema de vuelo de un avión o el control de velocidad de un tren de alta velocidad puede poner en grave riesgo la vida de muchas personas. Bajo un enfoque de tiempo real crítico, los sistemas de memoria basados en cache no son los más adecuados, dado que no garantizan el rendimiento en la ejecución del caso peor, incorporando impredecibilidad al sistema [35]. La mejora en el rendimiento medio que proporciona la memoria cache no es una opción aceptable para un sistema en tiempo real, donde un pequeño retardo puede provocar el fallo del sistema. Sin embargo, sí son adecuados para sistemas en tiempo real no críticos, donde algo de indeterminismo tiene como resultado una degradación del rendimiento del sistema. El compromiso de calidad debe determinar el nivel de degradación aceptable para que se cumpla con la calidad de servicio.

Los sistemas empotrados, no diseñados para sistemas en tiempo real, han ido incrementando su presencia en los dispositivos móviles de última generación que han sufrido una gran evolución en las últimas décadas, y que nos proporcionan acceso a la información de forma ubicua. El número y tipo de aplicaciones soportadas se ha incrementado drásticamente con aplicaciones multimedia, de reconocimiento facial o de voz, renderización de imágenes, conectividad de red, etc. y con ello la com-

plejidad de ejecución en entornos más restrictivos que los sistemas de sobremesa o portátiles.

Estas aplicaciones, que incorporan interfaces gráficas y trabajan con algoritmos más complejos que las aplicaciones tradicionalmente soportadas por los sistemas empotrados, requieren mayor rendimiento y restricciones de tiempo distintas. Hay que tener presente que, generalmente, estos dispositivos son de reducido tamaño y dependen para su funcionamiento de una batería. Por otra parte, la potencia consumida afecta al tiempo de vida de la batería y es un factor crítico que incide directamente en el rendimiento del sistema. Las operaciones de audio y vídeo, presentes en las aplicaciones multimedia, necesitan de unos claros compromisos temporales. Por tanto, las aplicaciones multimedia deben cumplir con los compromisos de robustez, estabilidad y predecibilidad. Todos estos requisitos incrementan la presión sobre los componentes de los sistemas empotrados, entre ellos el subsistema de memoria. Una de las características de estas aplicaciones es el uso intensivo de datos, lo que se traduce en accesos continuos a memoria, y esto a su vez supone un gran impacto en aquellos sistemas empotrados con suministro de potencia mediante baterías. De hecho, la búsqueda del equilibrio entre coste, rendimiento, consumo de energía y disipación de potencia, manteniendo la eficiencia es uno de los mayores desafíos y un campo abierto a la optimización.

Otro factor a tener en cuenta, es el cambio constante que sufre el mercado, donde los tiempos de desarrollo son más cortos que en décadas pasadas y donde la flexibilidad de configuración y la eficiencia son factores muy importantes para cumplir con los compromisos de calidad. Dentro de las medidas que permiten evaluar la eficiencia, la minimización del consumo de energía es crucial para mantener los dispositivos a una temperatura razonable, con las exigencias de cómputo de las nuevas aplicaciones soportadas. En este entorno, el sistema de memoria ha sido señalado como un componente esencial para mejorar el rendimiento y el consumo de energía [36].

El problema de la optimización del subsistema de memoria es uno de los más

abordados en las últimas décadas, debido a la importancia que tiene dentro de los sistemas empotrados. La memoria es limitada, particularmente en sistemas donde la memoria y el procesador comparten el mismo chip, dispositivos *System on Chip* (SoC). Tal y como se ha mencionado anteriormente, el subsistema de memoria cache mejora el rendimiento medio y si bien no es adecuado para su implantación en un RTS crítico, en dispositivos como tabletas o teléfonos móviles, entre otros, se han ido incorporando y han permitido incrementar el rendimiento. En esta tesis se estudian varios niveles del subsistema de memoria, bancos de registros del procesador, gestión de memoria dinámica y se hace especial énfasis en el subsistema de memoria cache y en la flexibilidad de su configuración. Entre los objetivos se encuentra la reducción del consumo energético, lo que incidirá positivamente en el rendimiento, en el tiempo de vida de la batería y finalmente en la durabilidad del dispositivo. En la siguiente sección se realiza una revisión de la literatura relacionada con los temas que se abordan en esta tesis.

2.2. Estado del arte

El incremento de la densidad de potencia de los microprocesadores, gracias a los avances en la tecnología de fabricación de circuitos y a la necesidad de mejorar el rendimiento, ha contribuido al incremento del problema térmico y a que este se convierta en una de las principales áreas de desarrollo e investigación. El diseño de los procesadores de los nuevos sistemas y particularmente de los sistemas empotrados ha sufrido cambios provocados, entre otros, por el impacto que las altas temperaturas tienen en el consumo de energía, fiabilidad y coste de los sistemas de enfriamiento. Un incremento de la temperatura del chip entre 10 y 15^{o}C puede reducir su tiempo de vida casi a la mitad [37]. La reducción del calentamiento y del consumo térmico, para mejorar la seguridad y fiabilidad de los sistemas actuales, ha sido abordada por los investigadores desde diferentes perspectivas, las más relevantes se mencionarán a lo largo de la sección.

El subsistema de memoria, tal y como se ha mencionado anteriormente, es un

importante cuello de botella en el sistema, dado que no puede competir con la velocidad del procesador y la diferencia entre ambos continúa incrementándose. Este problema convierte al subsistema de memoria en uno los elementos más importantes desde el punto de vista del diseño de sistemas empotrados, debido a las restricciones a las que los diseñadores deben enfrentarse. Ocupando un espacio significativo dentro del chip y con un consumo superior a la mitad de la energía total consumida por el procesador [38, 39], es uno de los elementos que más contribuye al consumo total de energía, por tanto es uno de los que más pueden ayudar a reducirlo. Así, su optimización ha sido el objetivo de multitud de trabajos de investigación y por tanto, ha contribuido al desarrollo y la innovación tecnológica.

El comportamiento del sistema de memoria afecta tanto al rendimiento como al consumo de energía. Conseguir el equilibrio entre el incremento del rendimiento y la reducción del consumo energético a un costo razonable, es uno de los principales retos en el diseño de sistemas empotrados. En las últimas décadas, han surgido técnicas de optimización hardware, software y combinaciones de ambas, que han permitido incrementar la eficiencia de estos sistemas.

En la Figura 2.7 se muestra una organización típica de memoria para un sistema empotrado. En ella se identifican claramente los diferentes componentes hacia donde se han dirigido las propuestas de optimización durante años. Esta tesis se centra en varios de estos componentes, particularmente, en la gestión del banco de registros, de la memoria cache y de la memoria dinámica. Dada la amplitud de trabajos en la literatura, en esta tesis se mencionan algunos de aquellos que están más cercanos a los enfoques de optimización que se proponen.

2.2.1. Gestión del banco de registros

El banco de registros se identifica dentro del subsistema de memoria como uno de los grandes consumidores de energía [40]. Los cambios en las características de los microprocesadores llevan a que el banco de registros deba soportar accesos concurrentes. Por otra parte las aplicaciones multimedia ejercen mayor presión

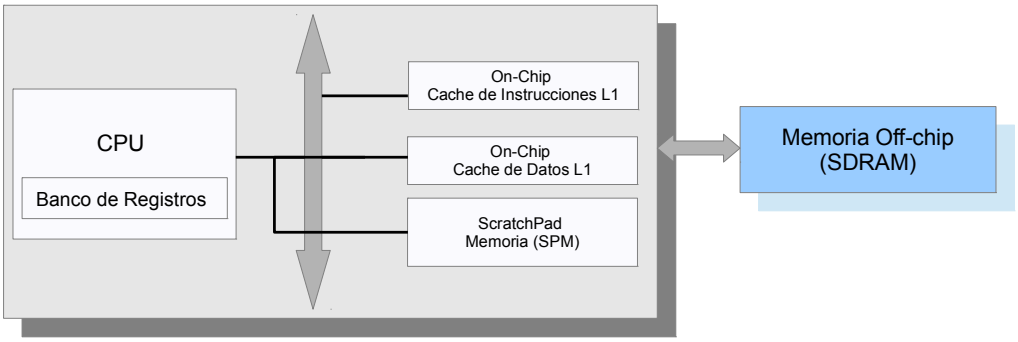


Figura 2.7: Esquema general de una organización típica de memoria en un sistema empotrado. El subsistema de memoria está compuesto por el núcleo del procesador (CPU) y el banco de registros, un sistema de memoria cache de nivel 1 y una memoria *scratchpad* dentro del mismo chip del procesador. Finalmente, la memoria principal que se encuentra fuera del chip del procesador. Otros elementos pueden conformar otro esquema típico, como una pequeña parte de la SDRAM que puede también estar on-chip, y a pesar de poder soportar varios niveles de cache, generalmente, la mayoría de los sistemas empotrados implementan un único nivel.

sobre el subsistema de memoria, principalmente por las características intrínsecas de la información que manejan. Esto conlleva un incremento en la frecuencia de intercambio de información entre el banco de registros y el siguiente nivel de la jerarquía de memoria. Por tanto, el número de accesos al banco de registros se incrementa significativamente provocando un incremento en cuanto a la necesidad de disipación de potencia, y por consiguiente en la temperatura del chip [8]. Un menor consumo de energía permite reducir la disipación la potencia y la temperatura, para de ese modo, aumentar la fiabilidad y el rendimiento del sistema. Este problema se ha abordado proponiendo diferentes técnicas de optimización bajo perspectivas software y hardware, como a continuación se enumeran.

En 2003, Ayala et al. [41] presentan la implementación de un mecanismo hardware en procesadores con ejecución fuera de orden con el que obtienen una mejora del 60 % de la energía consumida por el banco de registros. El enfoque modifica el algoritmo de renombramiento de registros, con una implementación separada de los búferes de renombramiento para realizar la asignación de registros. El algoritmo permite desactivar los registros no utilizados y mantenerlos en un estado de bajo consumo, para ello se utilizan en la implementación celdas de memoria *drowsy*. Las

celdas *drowsy* [42] se utilizan como técnica en los circuitos para asignar diferentes niveles de voltaje y, por tanto, reducir el consumo. Así, cuando no es necesario acceder a un registro se le asigna el nivel de voltaje más bajo. Los registros requeridos para su uso se pueden activar por adelantado sin pérdida de rendimiento.

Posteriormente, los mismos autores en [43] analizan el comportamiento de la técnica de desenrollado de bucles, añadiendo una nueva modificación de la asignación de registros dirigida por el compilador. El objetivo es reducir el número de puertos del banco de registros, responsable del incremento del consumo de energía y del tiempo de acceso. Este trabajo utiliza la implementación realizada en [41], con una arquitectura con organización en bancos, simple y de baja potencia. El algoritmo de asignación modificado se encarga de que cada operando de una instrucción se asigne a un banco diferente. La consecuencia inmediata es la reducción del número de puertos gracias al aumento del paralelismo. La aplicación de la técnica de desenrollado de bucles se realiza previa estimación del número de registros a utilizar y de esta forma, se mantienen desactivados aquellos registros no requeridos.

Atienza et al. [44] se centran en las características de nuevos Procesadores Digitales de Señal (DSP, *Digital Signal Processing*) y Procesadores Digitales de Conjunto de Instrucciones de Aplicación Específica (ASIP, *Application-Specific Instruction-Set Processor*), diseñados para soportar especialmente los nuevos requisitos de las aplicaciones multimedia, en sistemas empotrados. Las arquitecturas DSP y ASIP tienen bancos de registros muy grandes y varios puertos que incrementan el consumo de energía. Los autores adecúan estas características a una arquitectura VLIW base, sobre la que se añaden extensiones como la organización en múltiples bancos de registros y la aplicación de la técnica de Escalado Dinámico de Tensión (DVS, *Dynamic Voltage Scale*), que permite mantener los bancos no utilizados en estado de bajo consumo. Se incorporan las optimizaciones dirigidas por el compilador proporcionadas en [41, 43], y se modifica el algoritmo de asignación de registros para que todos los operandos de una instrucción se asignen a registros pertenecientes a un mismo banco de registros, el resto se desactivan. El estudio pre-

senta una mejora del 60 % en el consumo de energía del banco de registros. Igual porcentaje de mejora es alcanzado posteriormente en [45] donde los autores aplican la extensión hardware DVS y mecanismos de desactivación de aquellos registros no utilizados, para optimizar el consumo de energía. Estos trabajos ponen su foco de atención en la optimización del consumo de energía y no particularmente en la temperatura.

La propuesta presentada por Zhou et al. en [46] tiene como objetivo minimizar la densidad de potencia y la temperatura pico. En una organización con múltiples bancos de registros, la implementación permite distribuir los accesos a registros dentro de cada banco de registros, sin traspasar los límites de cada uno. El proceso necesita de información recopilada de las aplicaciones sobre los accesos a registros y la frecuencia de ejecución. El proceso es llevado a cabo por el compilador, durante la fase de asignación de registros y tras la fase tradicional de asignación. Se presentan dos alternativas de optimización, la primera de ellas permite agrupar registros en particiones virtuales y realizar la reasignación de nombre de registro entre registros individuales de las particiones establecidas. La segunda analiza el intercambio de nombres entre dos intervalos de vida de registros, completo o a nivel de un subconjunto de los intervalos de vida de los registros a reasignar. Los resultados son bastante similares al aplicar la primera alternativa o ambas de forma conjunta, y obtiene una reducción de la densidad de potencia que varía entre 4^{o}C y 7^{o}C, con respecto a las técnicas de optimización clásicas por parte del compilador. Sin embargo, la optimización estaba diseñada para un conjunto reducido de arquitecturas VLIW o RISC.

Otras técnicas de optimización térmica, dirigidas por el compilador, incrementan el paralelismo a nivel de instrucción mediante una mejora en la planificación de las instrucciones a ejecutar [47]. El resultado obtenido es una reducción del número de accesos al banco de registros, del consumo de energía y de temperatura.

Recientemente, Sabry et al. [48] proponen un nuevo flujo de compilación para minimizar el problema térmico del banco de registros y reducir los puntos calien-

tes, minimizando la difusión mutua y el efecto de autocalentamiento. Aunque puede ser implementado para varias arquitecturas, su objetivo principal son las arquitecturas basadas en ventanas de registros y aquellas con una organización en bancos. De acuerdo a los datos publicados, este nuevo enfoque consigue una reducción del 91 % de los puntos calientes, y del 11 % de las temperaturas media y pico. Con objetivos muy próximos al estudio mencionado anteriormente, Sharifi et al. [49] plantean una gestión térmica y del consumo de energía de forma conjunta en sistemas multiprocesador on-chip (MPSoC, *Multiprocessor System-on-Chip*), especialmente para sistemas empotrados. Un análisis off-line permite identificar las necesidades de rendimiento de la tarea actual. Posteriormente, con la planificación de tareas y la técnica DVS se intenta llegar al compromiso de rendimiento, a la vez que se minimiza el consumo de energía y el desequilibrio térmico. Los autores presentan la comparación respecto a otras técnicas de balanceo de carga, con un mejora del 11 % en el consumo de energía, a la vez que reducen los puntos calientes y la temperatura.

Desde un punto de vista teórico, Brooks et al. [50] hacen una recopilación de diferentes modelos desde los que analizar de forma precisa y eficaz el problema térmico (dinámico o en estado estacionario, análisis de la potencia disipada) y de consumo de energía (analíticos y empíricos), que afectan a la fiabilidad del sistema. De forma detallada, se argumenta la necesidad de realizar análisis exhaustivos que identifiquen los puntos críticos en cuanto a las propiedades térmicas del diseño arquitectónico de microprocesadores.

Los cambios en la tecnología de fabricación y el incremento de las necesidades de rendimiento de las aplicaciones modifican los requisitos de diseño. La relación existente entre consumo de energía dinámica, la producción de calor y el incremento de temperatura en el procesador se presentan como uno de los principales problemas a resolver. Los cambios producidos en el consumo de potencia pueden derivar en fallos temporales, en la necesidad de reducir la frecuencia de trabajo del procesador, en detrimento del rendimiento, e incluso puede llevar a fallos permanentes del microprocesador. Sin olvidar que las baterías que alimentan, por ejemplo, los

dispositivos móviles, pueden ver limitado su tiempo de vida. Por tanto, la fiabilidad del sistema puede verse en entredicho, además del coste que supone un sistema de enfriamiento eficiente. Teniendo presentes estas consecuencias, la identificación temprana de las puntos calientes, en las fases de diseño y modelado previas al proceso de implementación y bajo diferentes escenarios se consideran fundamentales para obtener un diseño eficiente.

Una fase de análisis suele llevar implícito un alto coste computacional debido a que normalmente se utilizan métodos numéricos. Generalmente, los modelos analíticos y empíricos de potencia han sido utilizados para analizar la eficacia energética de las diferentes arquitecturas. Los modelos analíticos son un método rápido en la estimación del consumo de potencia, sin embargo cuando se depende del diseño físico del circuito las estimaciones pueden no ser muy precisas. Los modelos empíricos se basan en el análisis de la potencia a nivel del circuito y son muy útiles cuando los diseños sufren pequeñas diferencias con el original, aunque el proceso de simulación es lento. El modelo empírico es utilizado por PowerTimer [51] que estudia la potencia de los microprocesadores según el comportamiento de los bloques funcionales, en los que se divide la microarquitectura. El modelo analítico es utilizado, por ejemplo en [52] para analizar el impacto térmico en estructuras como el sistema de memoria cache o el banco de registros, entre otras.

En el Capítulo 4 se realiza un estudio analítico para ver la influencia de la temperatura, debida a los accesos al banco de registros, sobre la temperatura del procesador. El enfoque se dirige al banco de registros de las arquitecturas VLIW y ARM. Tras el análisis de los datos, se propone la ejecución de un algoritmo de optimización, basado en técnicas metaheurísticas, que utiliza la permutación de los accesos al banco de registros, de tal manera que aquellos registros que soportan muchos accesos queden alejados entre sí y permita reducir el incremento de temperatura.

2.2.2. Gestión de la memoria cache

A continuación de los registros del procesador, en el siguiente nivel de la jerarquía de memoria, se encuentra la memoria cache. Investigaciones como la realizada en [53] indicaban que la mitad de la energía consumida en un sistema empotrado se debe al subsistema de memoria. Además, identifican a la memoria cache como un cuello de botella en la energía del sistema y ponen el foco de atención sobre ella como firme candidata a optimizar. Estudios más recientes apuntan a la memoria cache on-chip como uno de los componentes que más energía consume, oscilando entre un 20 % y 30 % del total de energía del chip en un procesador empotrado [5]. Además, el diseño de la memoria cache tiene una gran influencia sobre el consumo de energía y el rendimiento, debido a la complejidad que implica las diferencias del hardware. Por ejemplo, una cache directamente mapeada presenta los tiempos de acierto más rápidos, pero necesita mayor tamaño de cache para obtener un rendimiento global comparable a una cache asociativa por conjunto de *N*-vías. Por otra parte, las caches asociativas por conjuntos presentan un mayor consumo energético dado que acceden a un conjunto de *N* etiquetas en cada operación [10]. De hecho, el comportamiento de la memoria cache afecta tanto al rendimiento como al consumo de energía.

El comportamiento de la memoria cache no está sólo condicionado por los parámetros estructurales (tamaño de cache, tamaño de bloque, asociatividad, etc.), sino también por parámetros relacionados con los algoritmos de búsqueda, prebúsqueda y reemplazo, o políticas de escritura, entre otros. Todas estas características forman la denominada configuración de cache. Encontrar los valores óptimos para estos parámetros llevará a mejorar el rendimiento, consumo de energía y, por consiguiente, a identificar la mejor configuración de cache. Además, las aplicaciones tienen comportamientos diferentes, distintos patrones de acceso a memoria [10] e incluso diferentes requisitos en cuanto a la configuración de la cache, para satisfacer los objetivos de consumo energético y rendimiento. Sin embargo, encontrar una configuración óptima de cache para una única aplicación no es la mejor opción.

Una configuración de cache que se ajusta a las necesidades de una aplicación determinada, es posible que no se ajuste a otra aplicación que tiene un patrón distinto de acceso a memoria. Así, el problema que se presenta es encontrar la configuración óptima de cache para el conjunto de aplicaciones ejecutadas en un dispositivo empotrado, lo cual no solo mejorará el rendimiento o el consumo de energía, sino también la fiabilidad a largo plazo. Afortunadamente, los sistemas empotrados están especialmente diseñados para un conjunto limitado de aplicaciones. Varias propuestas de optimización, bajo diferentes enfoques, han sido dirigidas a incrementar el rendimiento y/o reducir el consumo energético, al desarrollo de herramientas de simulación y evaluación, etc. A continuación se analizan algunos de los trabajos más relevantes que han abordado la optimización de la memoria cache.

2.2.2.1. Reconfiguración de cache

Uno de los objetivos durante años ha sido configurar la memoria cache de acuerdo a la carga de trabajo que se ejecuta para mejorar el rendimiento y/o el consumo de energía. Esto requiere tanto soporte hardware para la reconfiguración de cache como algoritmos para decidir los valores adecuados de los parámetros que obtengan el mejor rendimiento teniendo en cuenta la aplicación en ejecución. En este sentido, muchos trabajos de investigación han analizado los parámetros de la memoria cache y han propuesto nuevas técnicas de reconfiguración desde diversos puntos de vista.

Malik et al. en [53] analizan el comportamiento del sistema respecto del consumo de energía con un tamaño de cache fijo y un conjunto muy limitado de parámetros. Sobre una memoria cache unificada de nivel 1, se modifican los parámetros configurables como la asociatividad, modo de operación de escritura (post-escritura [4] y escritura inmediata [5]) y utilización de búferes, sobre un conjunto de aplicaciones. La conclusión a la que llegan los autores es que las mejoras obtenidas en rendimiento y consumo de energía dependen de la configuración propuesta. La

[4] *copy-back* o post-escritura: los datos se actualizan en la cache y sólo cuando el bloque va a ser reemplazado, se actualiza en la memoria principal.

[5] *write-through* o escritura inmediata: los datos se actualizan tanto en la cache como en la memoria principal.

mejor configuración de cache será aquella que minimice el tiempo de ejecución y consumo energético.

En cuanto al rendimiento, varios trabajos de investigación han sido desarrollados con el objetivo de mejorar el rendimiento cambiando los parámetros estructurales de las memorias cache. Por ejemplo, en [54], las aplicaciones son analizadas durante determinados intervalos o fases de la ejecución. El objetivo es clasificar los patrones de comportamiento y aplicar diferentes configuraciones de cache en cada caso. El enfoque presentado en esta tesis es diferente, dado que se intenta obtener el comportamiento de un conjunto de aplicaciones, obteniendo su perfil estático completo. Es decir, cada aplicación es simulada completamente o hasta un número representativo de instrucciones y durante dicho proceso su comportamiento es recogido en un fichero para ser tratado posteriormente.

La optimización de la memoria cache, en cuanto al tiempo de ejecución, consumo energético o ambos, se ha abordado usando principalmente dos técnicas diferentes: reconfiguración dinámica y perfil estático.

Reconfiguración dinámica. Por lo que se refiere a la reconfiguración dinámica, Albonesi [1] presenta una técnica donde la cache se redimensiona adaptando la asociatividad de acuerdo a las necesidades de la aplicación objetivo. Este trabajo se ha convertido en referente y en la base de otros muchos desarrollados posteriormente. Por ejemplo, Zhang et al. [55] desarrollan una técnica denominada *line-concatenation* que permite cambiar el tamaño de bloque. Posteriormente, en [56] implementan una técnica denominada *way-concatenation* para adaptar el tamaño de la cache, donde se utiliza el mecanismo de desactivación de vías y así, mejorar el ahorro de energía. Ambos enfoques son propuestos en [57] con el objetivo de conseguir una configuración de cache que reduzca el consumo de energía, adaptando el tamaño total de la cache y el tamaño de bloque de la cache vía software. En [58] la técnica conocida como *way-concatenation* es utilizada para reconfigurar la cache de sistemas empotrados mediante software y así, minimizar el consumo de energía.

Capacidad, asociatividad (directamente mapeada, 2-way y 4-way) y tamaño de bloque (16, 32 y 64) son los parámetros arquitectónicos estudiados.

Naz et al. [59] proponen un diseño con cache de datos dividida para sistemas empotrados. La cache de datos se divide según que el tipo de datos sea escalar o matricial para aprovechar la localidad temporal y espacial, respectivamente. Su optimización se basa en trabajar con configuraciones predefinidas de cache, donde los valores de los parámetros se fijan antes de la optimización.

Chen y Zou [60] presentan un algoritmo eficiente de gestión de reconfiguración en sistemas empotrados para mejorar el rendimiento y la disipación de energía. La configuración de la cache cambia cuando el algoritmo detecta de forma automática un cambio de fase y tras seleccionar la configuración óptima de cache en el espacio de búsqueda. Para optimizar la búsqueda de la configuración óptima de cache, esta se divide en varias particiones de diferentes tamaños, donde sólo la partición seleccionada está activa, el resto permanece desactivada para ahorrar energía. El espacio de búsqueda está formado por el tamaño de cache, tamaño de bloque y asociatividad.

Del mismo modo, Gordon-Ross et al. [61] presentan un algoritmo eficiente de gestión de reconfiguración dinámica para sistemas empotrados, donde la configuración cambia automáticamente después de identificar un cambio de fase. El espacio de búsqueda está compuesto por el tamaño de cache (2, 4, u 8 KBytes), tamaño de bloque (16, 32, or 64 bytes) y asociatividad (1-way, 2-way or 4-way). El intervalo definido para realizar el ajuste de la cache varía según la información de retroalimentación recibida por el algoritmo. Este trabajo se amplió a un segundo nivel de cache en [62].

López et al. [63] proponen un algoritmo on-line sobre sistemas SMT (*Simultaneous Multithreading*), capaz de determinar la mejor configuración de cache en tiempo de ejecución para un conjunto de trabajo dado. El algoritmo de toma de decisión se ejecuta tras un conjunto fijo de instrucciones. A partir de la información obtenida en el intervalo anterior se predice el comportamiento de la cache para el

intervalo siguiente. La técnica es similar a la de control predictivo propuesta por [1]. Sin embargo, este trabajo no está dirigido a sistemas empotrados.

Posteriormente, Gordon-Ross et al. [64] aplican un método con una fase off-line de clasificación y una fase on-line de predicción. La fase de clasificación separa la ejecución de la aplicación en intervalos de tamaño fijo y los agrupa por similitud según el patrón de comportamiento. La fase de predicción decide la configuración de cache a aplicar en el siguiente intervalo. El espacio de exploración está definido por el tamaño de cache, tamaño de bloque y asociatividad, lo cual significa que sólo se consideran 3 parámetros para definir el espacio de búsqueda.

En [65, 66], se propone un enfoque de reconfiguración dinámica en sistemas empotrados en tiempo real no críticos (blandos), sobre una jerarquía de memoria cache de un nivel y multinivel, respectivamente. En ambos trabajos se utiliza una combinación de análisis estático y dinámico para adaptar los parámetros de cache, sin que se produzca sobrecoste respecto al tiempo de ejecución.

Recientemente y tomando como base los trabajos mencionados anteriormente, Wang et al. en [67] presentan un método de reconfiguración dinámica de memoria cache en sistemas empotrados en tiempo real no críticos. Utilizan un análisis estático en tiempo de ejecución que les permite minimizar el consumo de energía. El proceso de análisis identifica la mejor configuración de cache para cada una de las fases, en las que se dividen las tareas en ejecución. No obstante, en todos ellos, el número de parámetros a optimizar es reducido y está compuesto por el tamaño de cache (1, 2 o 4 KB), tamaño de bloque (16, 32 o 64 bytes) y asociatividad (1-way, 2-way o 4-way).

Con el objetivo de reducir el consumo de potencia, Huang et al. [68] proponen una nueva arquitectura con una cache L3 y un diseño formado por elementos pequeños independientes. Partes de esta memoria pueden ser deshabilitadas para utilizar un modo "*stand-by*" cuando no se están usando. Si se necesita una parte deshabilitada, se activa sin sobrecarga de rendimiento. En este trabajo, se reduce el consumo de energía entre un 20 % y 40 % trabajando a altas y bajas frecuencias,

respectivamente. Los parámetros analizados son el tamaño de cache (1, 2, 4 y 8 KB), asociatividad (1, 2, 4 vías) y tamaño de bloque (16, 32, 64 bytes).

Gracia et al. [69] estudian el modo de acceso a cache en mosaico e identifican cuáles son la fuentes que derrochan energía en LP-NUCA (accesos en paralelo a las etiquetas, matrices de datos, fases con baja localidad, migración de bloques inútiles) y proponen un controlador (ADR, *Adaptive Drop Rate*), basado en *hill climbing*[6], para reducir la tasa de reemplazos y la migración de bloques en el caso de baja localidad. Los autores indican que este controlador no sólo es válido para LP-NUCA. El ahorro energético llega al 29 % y la reducción de migraciones llegan al 81 %. Sin embargo, este enfoque difiere significativamente del que se propone en esta tesis.

Las nuevas tecnologías hardware y la tecnología multicore, como las propuestas, por ejemplo, por ARM [70], permiten adaptar la configuración de cache a la aplicación en ejecución. Los cambios afectan a los principales parámetros como el tamaño de la cache, tamaño de bloque y asociatividad. No obstante, es necesario realizar una exploración exhaustiva para determinar cuales son los valores óptimos de cada aplicación.

Los trabajos mencionados anteriormente minimizan el tiempo de ejecución o el consumo energético seleccionando los parámetros de cache que obtengan el mejor rendimiento para cada aplicación en ejecución. El principal inconveniente de la reconfiguración dinámica es que añade complejidad adicional al diseño del subsistema de memoria y en la mayoría de ellos, esta complejidad añade una sobrecoste en cuanto a tiempo de ejecución. Además, en el caso de la ejecución concurrente de varias aplicaciones que comparten la memoria cache, el mecanismo de reconfiguración podría necesitar ser llamado cada vez que se produce un cambio de contexto y, de esta manera, acumular los sobrecostes de tiempo asociados.

Perfilado estático. Con respecto al uso de perfilado estático, Ghosh et al. presentan en [71] un modelo de cache analítico combinado con un algoritmo que estima

[6] *Hill climbing* es una técnica matemática de optimización perteneciente al conjunto de técnicas de búsqueda local.

la configuración de cache que debe satisfacer las restricciones de rendimiento con respecto a los fallos de cache. Sin embargo, sólo optimizan tamaño de cache y asociatividad.

Li aplica en [72] un nuevo método para acelerar la evaluación de diferentes configuraciones de cache, en el diseño de sistemas empotrados. El método realiza perfilado estático mediante la compresión de las trazas de las aplicaciones, y así reducir el proceso de evaluación de cache. Este enfoque, no obstante, sólo optimiza el parámetro de asociatividad.

Gordon et al. [73] presentan técnicas estáticas para explorar posibles configuraciones de cache para una aplicación completa o una única fase de la ejecución. Janapsatya et al. en [74] proponen un algoritmo de bosque modificado que utiliza estructuras de datos simplificadas para encontrar la mejor configuración de cache para una aplicación dada. Los autores optimizan los parámetros estructurales básicos de tamaño de cache, tamaño de bloque y asociatividad.

La propuesta presentada por Andrade et al. en [75] es una extensión de una técnica de modelado analítico sistemático basada en ecuaciones probabilísticas de error, que permite el análisis automático del comportamiento de la cache en códigos con patrones irregulares, resultado de indirecciones. Sin embargo, estos modelos sólo pueden optimizar el tamaño de cache y la asociatividad, a diferencia del enfoque que se propone en esta tesis.

Rackesh Reddy en [76] estudia el efecto de las cargas de trabajo multiprogramadas sobre la cache de datos en un entorno multitarea apropiativa y propone una técnica que mapeando tareas a diferentes particiones de cache, reduce significativamente tanto la energía dinámica como la pérdida de energía.

Xingyan y Hongyan [77] se basan en un esquema de perfiles del algoritmo de reemplazo de cache OPT (algoritmo óptimo de reemplazo), para presentar un método que genera las mejores sugerencias de cache estáticas. De igual forma, Feng et al. [78] aplican una nueva política de reemplazo para llevar a cabo la toma de decisión basada en la reutilización de la información de las líneas de cache y los datos

solicitados, desarrollando dos tipos de predictores de reutilización de la información: un predictor estático basado en perfiles y un predictor en tiempo de ejecución. Sin embargo, estos dos últimos enfoques sólo mejoran el algoritmo de reemplazo. Gordon-Ross et al. [79] estudian la interacción entre código de reordenamiento y la configuración de cache, obteniendo excelentes resultados. Sin embargo, esta técnica se aplica sólo a la cache de instrucciones.

Las propuestas presentadas en esta tesis se basan en una estrategia estática utilizando algoritmos de optimización off-line que evita la reconfiguración de cache según las aplicaciones que puedan ejecutarse concurrentemente y que no añade complejidad hardware adicional al diseño del subsistema de memoria estándar. Los enfoques se centran en encontrar la mejor configuración de cache para un conjunto de aplicaciones después de explorar el espacio de búsqueda de los diseños de cache. La mejor configuración de cache se define con los valores óptimos para el conjunto de parámetros a configurar de una memoria cache de nivel 1.

El espacio de diseño de los trabajos anteriores está definido por los parámetros de tamaño de cache, tamaño de bloque y asociatividad de la cache de instrucciones o datos, aunque no se ha encontrado ninguna propuesta que considere ambas caches (datos e instrucciones), al mismo tiempo. Algunos de ellos se centran en la política de reemplazo o no están dirigidos a sistemas empotrados. El enfoque que se propone en esta tesis aborda de forma conjunta la cache de instrucciones y de datos, y se presentan como parámetros a optimizar: tamaño de cache, tamaño de bloque, asociatividad como parámetros arquitectónicos; algoritmo de reemplazo y algoritmo de prebúsqueda para ambas caches y política de escritura para la cache de datos. Por tanto, el espacio de búsqueda es mayor que el que se encuentra en la literatura.

Técnicas evolutivas. Las técnicas evolutivas han sido aplicadas ampliamente en la optimización del diseño, tanto optimización hardware como software con diferentes objetivos. Palesi y Givargis [80] presentan un enfoque de optimización multi-

objetivo para explorar el espacio de diseño de una arquitectura SoC y encontrar las configuraciones óptimas de Pareto del sistema. Se define una configuración óptima de Pareto como aquella configuración donde se alcanza un equilibrio en la mejora de los objetivos definidos y para la cual no es posible encontrar una solución alternativa mejor. Es decir, la mejora de uno de los valores objetivos en las posibles soluciones alternativas implica la degradación de alguno de los objetivos restantes.

Filho et al. [81] presentan un enfoque basado en un algoritmo NSGA-II para evaluar configuraciones de cache sobre una cache de nivel 2 para mejorar el consumo energético y el rendimiento. Sólo los parámetros estructurales básicos como el tamaño de cache, tamaño de bloque y asociatividad se tuvieron en cuenta en la optimización.

Bui et al. [82] aplican técnicas de particionamiento para solucionar el problema de la interferencia de cache. Un algoritmo genético sencillo establece el tamaño de cada partición de cache. Las tareas se asignan a las particiones de tal manera que se reduce al mínimo la utilización del sistema en la ejecución del caso peor y mejora la planificación en tiempo real.

Dani et al. [83] aplican un algoritmo genético para encontrar una configuración óptima de cache para un sistema multiprocesador en chip (MPSoC). Tratan con un sistema complejo como es un MPSoC. El número de parámetros no es muy alto y dado que codifican la configuración tanto del chip como de la cache en un cromosoma, el proceso de decodificación es complejo.

Hasta donde sabemos, ninguno de los trabajos sobre memoria cache, mencionados anteriormente, considera la optimización del conjunto de parámetros que ha sido abordado en esta tesis. La mayoría trabaja sobre un espacio de búsqueda basado en los parámetros básicos estructurales de cache como son el tamaño de cache, tamaño de bloque y asociatividad. Además, los valores posibles para cada parámetro configurable son bastante reducidos a diferencia de los valores que se tienen en cuenta en esta tesis. Otro aspecto a tener presente es que la metodología que se propone puede ser aplicada a otros tipos de cache.

Por tanto, el espacio de búsqueda propuesto se define como todas las combinaciones posibles de los valores de esos parámetros. De esta forma, cuanto mayor es el número de configuraciones, mayor es la dificultad para realizar la exploración debido al crecimiento del espacio de búsqueda. La propuesta presentada en esta tesis se basa en el uso de técnicas heurísticas para abordar dicha complejidad.

2.2.3. Gestión de la memoria dinámica

Actualmente, tal y como ya se ha mencionado, las aplicaciones multimedia se ejecutan en una amplia variedad de plataformas hardware tales como ordenadores personales, consolas de videojuegos, smartphones y otros muchos dispositivos móviles. Estas aplicaciones normalmente llevan a cabo un elevado número de operaciones de memoria dinámica debido a la naturaleza inherente de los datos que manejan. De esta forma, el gestor de memoria dinámica, también conocido como *Memory Allocator*, es un componente crucial que influye en el comportamiento medio de la aplicación y del hardware. Por tanto, dos cuestiones importantes son saber cómo medir la influencia de los DMM en el comportamiento de la plataforma hardware y aplicación, y poder evaluar de forma ágil las necesidades de una aplicación objetivo en cuanto a memoria dinámica y, de esta manera, poder diseñar DMM personalizados.

Berger et al. en [3] estudian el comportamiento de la asignación dinámica de almacenamiento, para gestores de memoria dinámica personalizados y de propósito general, para programas en C++. Los autores desarrollan una biblioteca de plantillas en C++ que se utiliza para implementar diferentes gestores de memoria, y para estudiar las métricas de tiempo de ejecución y uso de la memoria. Los resultados que presentan son bastante representativos. Sin embargo, es necesario un gran esfuerzo de codificación para extenderlos a otras aplicaciones o diseños de gestores de memoria. Además, este estudio no tiene en cuenta métricas tan significativas como el consumo de energía y la temperatura, que son verdaderamente importantes, especialmente para dispositivos móviles.

Lo et al. en [84] presentan un DMM que recibe trazas provenientes de programas desarrollados en Java y C++ y realiza la simulación de acuerdo a un esquema de asignación seleccionado por el usuario. El trabajo proporciona algunas técnicas para generar trazas y el simulador presenta un conjunto de métricas relativas al tiempo de ejecución y uso de la memoria. Teng et al. en [85] presentan un enfoque similar. Sin embargo, ambos se limitan a un conjunto reducido de esquemas de asignación y el mejor DMM se selecciona en base a un conjunto de reglas predefinidas y búsqueda mono-objetivo. Además, tampoco se analiza el consumo energético y la temperatura.

Del Rosso en [86] evalúa el rendimiento de diferentes DMMs para sistemas empotrados en tiempo real. Este trabajo se centra en el estudio de la fragmentación de memoria, que es la principal medida para los sistemas empotrados en tiempo real. Además, estudia una métrica independiente de la CPU y distinta al tiempo de ejecución, denominada *performance speed metric*. Esta métrica evita la influencia del código de instrumentación en las restricciones de tiempo real. No obstante, este estudio se limita a sistemas empotrados en tiempo real y también adolece de medidas tan importantes como el consumo de energía y temperatura.

Más recientemente, Atienza et al. en [87] y [88] proponen un método para evaluar la memoria y la energía consumida por un DMM. Sin embargo, este trabajo necesita de la implementación e integración del DMM en la aplicación objetivo. Además, estos enfoques necesitan la ejecución de la aplicación cada vez que se quiere evaluar un gestor de memoria personalizado, y esta tarea consume mucho tiempo, particularmente si la aplicación requiere de interacción con el usuario, como sucede con los videojuegos.

Risco et al. [89] proponen un algoritmo evolutivo multi-objetivo paralelo para optimizar aplicaciones típicamente ejecutadas en sistemas de sobremesa, para su uso en sistemas empotrados multimedia. El objetivo era mejorar el proceso de asignación de tipos de datos dinámicos (DDT, *Dynamic Data Type*), que afecta al rendimiento. Este trabajo mejora el rendimiento medio, uso de la memoria y consumo

de energía del subsistema de memoria. Risco et al. en [90] presentan una primera propuesta de un algoritmo de optimización basado en GE, denominado GEA, con el fin de diseñar DMM personalizados y en [91, 92] se presenta y describe el simulador DMM que se utiliza en esta tesis. Sin embargo, el espacio de diseño se define según las clasificaciones realizadas en [93] y [87]. Esto conlleva un espacio de diseño muy grande y el proceso de clasificación da lugar a una taxonomía compleja de los DMM. Además, el tiempo de ejecución necesario para ejecutar el algoritmo y el simulador lo convierte en inaccesible y en [94] se presenta una versión paralela. Sin embargo, el perfilado de las aplicaciones se realiza sobrecargando las funciones *malloc()* y *free()* que requiere modificar y re-compilar cada aplicación y este proceso consume mucho tiempo.

En esta tesis, se parte de la primera propuesta de GEA y se presenta una nueva metodología de optimización multi-objetivo, basada en programación genética, que facilita la exploración y evaluación del espacio de diseño de los gestores de memoria dinámica y, por tanto, el diseño de gestores de memoria personalizados según la aplicación objetivo. La propuesta permite evaluar el comportamiento de cada DMM sin tener que modificar ni re-compilar la aplicación. El Capítulo 6 se dedica a explicar los detalles de la metodología y los resultados obtenidos.

En el siguiente capítulo se realiza una introducción a las técnicas metaheurísticas dado que son las técnicas seleccionadas para dirigir las propuestas de optimización en cada uno de los niveles de jerarquía de memoria abordados.

Capítulo 3

Heurísticas

Este capítulo revisa los fundamentos de los algoritmos utilizados en las propuestas de optimización desarrolladas. En primer lugar, se introducen los conceptos básicos de optimización, heurísticas, metaheurísticas y finalmente las técnicas de optimización evolutivas.

Optimización, en un problema de búsqueda, se puede definir como el proceso que permite explorar el conjunto de soluciones posibles, con el fin de encontrar la mejor, o una identificada como suficientemente buena al problema dado. Cuando el número de soluciones posibles a explorar es pequeño, encontrar la mejor solución se puede considerar como una tarea sencilla que puede ser resuelta, incluso de forma manual. Sin embargo, cuando el espacio de búsqueda aumenta considerablemente, se hace necesario el uso de técnicas avanzadas que orienten la búsqueda y permitan reducir el tiempo necesario para obtener la solución.

Una heurística es una técnica que se apoya en reglas que le ayudan en la búsqueda de soluciones buenas y viables con un coste computacional razonable. Se define una solución buena para un problema dado, como aquella que enfrentada a un valor de referencia mediante una función de evaluación, lo mejora. Sin embargo, para muchos problemas se desconoce el algoritmo exacto que permite obtener la solución óptima. En aquellos problemas donde el espacio de búsqueda es pequeño, la solución puede obtenerse mediante técnicas exactas como la búsqueda exhaustiva. Sin embargo, en estos métodos clásicos el tiempo de resolución de un problema crece en algunos casos exponencialmente, cuando hay que hacer frente a proble-

mas complejos de optimización y con grandes espacios de búsqueda. De hecho, el comportamiento que presentan empeora conforme el espacio de búsqueda se incrementa. Una limitación adicional se puede dar, si tras alcanzar un óptimo local, la técnica de optimización seleccionada no implementa ningún mecanismo que permita explorar otras zonas del espacio de soluciones. Por ello, las técnicas clásicas que no permiten seguir explorando el espacio de búsqueda tras encontrar un óptimo local, no son adecuadas para la resolución de este tipo de problemas. Para evitar el problema de los óptimos locales se combinan las técnicas heurísticas clásicas con algoritmos inteligentes y reciben la denominación de metaheurísticas [95].

Las técnicas metaheurísticas [95] se definen como "una clase de métodos aproximados que están diseñados para resolver problemas difíciles de optimización combinatoria en los que los métodos heurísticos clásicos no son efectivos. Las metaheurísticas proporcionan un marco general para crear nuevos algoritmos híbridos combinando diferentes conceptos derivados de la inteligencia artificial, la evolución biológica y los mecanismos estadísticos".

Durante mucho tiempo un amplio número de problemas no pudieron ser abordados por el alto coste computacional que ello suponía, problemas de optimización combinatoria, de diseño y control de dispositivos, procesos de aprendizaje automático, etc. Las metaheurísticas, con la combinación eficiente de diferentes metodologías, se adaptan al problema objetivo y permiten abordar problemas con espacios de búsqueda muy amplios. Dentro de los algoritmos metaheurísticos se encuentran métodos basados en trayectoria, como la búsqueda local iterativa (ILS, *Iterated Local Search*) [96], una variante de la búsqueda local clásica, o la técnica de enfriamiento simulado (SA, *Simulated Annealing*) [97]. Por otra parte, se encuentran los métodos basados en población como el método de optimización en colonias de hormigas (ACO, Ant Colony Optimization) [98], y los algoritmos evolutivos (EA, *Evolutionary Algorithm*) [99], ambos dentro de los denominados algoritmos bioinspirados. Esta sección se centra en el campo de los algoritmos evolutivos, aunque se puede encontrar información más detallada sobre las diferentes metaheurísticas

y su clasificación en [100, 101].

Las técnicas de Computación Evolutiva (EC, *Evolutionary Computation*), son una parte de la Inteligencia Artificial para la resolución de problemas de optimización [102]. Se basan en los mecanismos de la evolución natural y se utilizan ampliamente y con éxito en la resolución de problemas complejos, una gran parte de ellos problemas de optimización. La EC parte de un conjunto de posibles soluciones al problema dado, sobre las cuales se aplican técnicas de selección, generación de nuevas soluciones candidatas y de recombinación de características de las soluciones seleccionadas. En este capítulo se realiza un recorrido por los principales paradigmas de EC y las diferentes heurísticas empleadas en el desarrollo de los enfoques aquí propuestos.

3.1. Algoritmos evolutivos

Un Algoritmo Evolutivo se define como un método de optimización y búsqueda para la resolución de problemas combinatorios, basado en los procesos de la evolución natural.

Los algoritmos evolutivos trabajan sobre un conjunto de soluciones a un problema dado, y utilizan un conjunto de técnicas para seleccionar las mejores soluciones probables y mediante la recombinación de sus características, generar nuevas soluciones candidatas.

En la Figura 3.1 se muestra el esquema general de un EA sobre una población P, donde P_0 es la población inicial y P_t es la población en la generación t. Se parte de la población inicial, a partir de la cual se inicia el proceso de evolución representado en la Figura 3.1 por un bucle cuya condición de finalización depende, generalmente, del número de generaciones parametrizadas. Los individuos de la población se evalúan según una función de aptitud apropiada para el problema a resolver. A continuación se aplican los denominados operadores genéticos de selección, cruce y mutación. El operador de selección favorece la selección de los individuos más capacitados para la reproducción. A los descendientes se les aplica los operadores de

cruce y mutación sobre alguno de sus genes, en busca de que en la siguiente generación la adaptación de los individuos sea mejor que los individuos de la generación anterior. Finalizado el proceso evolutivo, el algoritmo devuelve los individuos mejor adaptados.

```
t=0;
Inicializar(P0)
Mientras (Condición de parada == falso)
  Evaluar (Pt)
  Progenitores = Seleccionar (Pt)
  Hijos = Recombinar (Progenitores), según probabilidad de cruce.
  Hijos = Mutar con cierta probabilidad
  Población nueva = Reemplazar (Hijos, Pt)
  t=t + 1
  Pt = Población nueva
fin
Retornar mejor solución encontrada
```

Figura 3.1: Esquema general de un Algoritmo Evolutivo. El algoritmo comienza con una población inicial P_0. A continuación y hasta alcanzar la condición de parada, los individuos son evaluados de acuerdo a una función de coste que depende del problema a resolver. El operador de selección se aplica a la población de progenitores P_t, para $t = 0, 1, ...g$, (siendo g el número de generaciones), para obtener los individuos mejor capacitados para sobrevivir y reproducirse. Mediante el operador de recombinación o cruce se obtiene la nueva población de descendientes sobre los que se aplica el operador de mutación que permite mejorar la diversidad de la población y obtener la nueva población P_t para $t = t + 1$. Tras alcanzar la condición de parada el algoritmo devuelve la mejor solución encontrada.

Para que un EA obtenga una muestra representativa de la población, esta debe tener una amplia diversidad genética, la cual viene dada por la diferencia entre los cromosomas de los individuos que conforman la población. Si no existe diversidad genética se tiende a un proceso de convergencia rápido y a que se obtenga un óptimo local que no represente el mejor individuo del espacio de búsqueda. El proceso de búsqueda se convierte en algo esencial en los EAs y, por tanto, los conceptos de exploración y explotación asociados a todo espacio de búsqueda juegan un papel crucial a la hora de definir la forma de trabajar de un EA. Los EAs necesitan buscar el equilibrio entre estos dos aspectos significativos, para aportar eficiencia al algoritmo. La búsqueda de dicho equilibrio es lo que se denomina ajuste de parámetros.

3.2. Paradigmas de la computación evolutiva

En 1993 se establece por primera vez la terminología de Computación Evolutiva. Sin embargo, las primeras referencias se remontan a finales de la década de los 50, cuando en [103, 104] se utilizan procesos evolutivos para resolver problemas mediante un sistema de computo, en concreto para programación automática. A pesar de ello, no es hasta la década de los setenta cuando se definen las principales técnicas de los Algoritmos Evolutivos: Programación Evolutiva (EP, *Evolutionary Programming*), Algoritmos Genéticos (GAs, *Genetic Algorithms*) y Estrategias Evolutivas (ES, *Evolutionary Strategies*).

En 1989 aparece la técnica denominada Programación Genética (GP, *Genetic Programming*), y como un caso especial de ésta última, aparece la técnica de Gramáticas Evolutivas (GE) en 1997. Los EAs proporcionan una gran robustez frente a problemas complejos y sobre grandes espacios de búsqueda, incluso trabajando con múltiples objetivos enfrentados entre sí, tal y como se presenta en la sección 3.3. Este comportamiento ha permitido asegurar su idoneidad para abordar con éxito problemas de optimización multi-objetivo. Así, los Algoritmos Evolutivos Multi-objetivo han ido paulatinamente incrementando su presencia, por ejemplo en aplicaciones de ingeniería medioambiental, naval, aeronáutica, electricidad y electrónica, robótica, etc. [105].

Por otra parte, algunos problemas relacionados con la ingeniería, operaciones financieras, etc. presentan ciertas características intrínsecas que dificultan su resolución haciendo uso de un enfoque analítico. De esta forma, la técnica de Evolución Diferencial [106] se presenta para abordar estos problemas complejos de optimización.

En esta sección se hace referencia a las características de los principales paradigmas de EC. Se prestará mayor atención y serán tratadas con más detalle aquellos directamente relacionados con esta tesis.

3.2.1. Programación evolutiva

La Programación Evolutiva fue concebida y desarrollada por Lawrence J. Fogel, quien implementa un algoritmo evolutivo sobre espacios de búsquedas discretos y a los cuales se aplica la operación de mutación. Un algoritmo básico de EP se muestra en la Figura 3.2. Fogel realizó experimentos que permitían evolucionar un autómata sobre una máquina de estados finitos [107, 108]. Asimismo, Fogel fue el primero en incorporar el concepto de población y las operaciones de selección y mutación como mecanismos de evolución de un autómata de estados finito. Estos experimentos contemplaban la inteligencia mediante un comportamiento adaptativo. Sus estudios se aplican con éxito a problemas de identificación, control automático, juegos y reconocimiento de patrones entre otros, pero sobre todo se aplican a problemas de predicción [109, 110].

```
1. Generar aleatoriamente la población inicial.
2. Evaluar la población inicial.
3. Seleccionar las mejores soluciones de la población inicial,
   generalmente se aplica el método de torneo.
4. Aplicar mutación para generar descendientes.
5. Evaluar población.
6. Seleccionar las mejores soluciones.
7. Si se han completado todos los pasos, entonces se finaliza, en
   caso contrario se vuelve al paso 4.
```

Figura 3.2: Algoritmo básico de Programación Evolutiva. El proceso comienza con la generación aleatoria de la población inicial. A continuación, la población es evaluada y se seleccionan las mejores soluciones (generalmente mediante torneo). La población de descendientes se obtiene tras aplicar el operador de mutación a los individuos seleccionados. La población de descendientes se evalúa según la función de coste definida y los mejores individuos se seleccionan para formar parte de la siguiente generación. El proceso se repite volviendo al paso 4 o finaliza si se ha alcanzado la condición de parada.

La programación evolutiva se concentra en la evolución a nivel de las especies, donde el operador de cruce no es necesario ya que se considera la premisa de que dos especies distintas no pueden ser cruzadas, a diferencia de otros algoritmos evolutivos.

3.2.2. Estrategias evolutivas

Peter Bienert, Ingo Rechenberg y Hans-Paul Schwefel, estudiantes de la Universidad Técnica de Berlín (Alemania), desarrollaron esta técnica en 1964 [111, 112]. Cada uno de ellos trabajaba de forma independiente, en el proceso de evolución que sufren los individuos en la naturaleza, mediante los mecanismos de mutación y recombinación de los genes de sus ancestros. La idea surgió frente a la complejidad de diseñar un sistema hidrodinámico, donde había que optimizar un conjunto de funciones, para las cuales no era posible utilizar las técnicas de optimización tradicionales. Las ES utilizan los operadores de cruce, mutación y selección (probabilístico o determinístico), donde los peores individuos se eliminan de la población.

La primera versión denominada $(1 + 1) - EE$, trabaja con un único padre y descendiente por generación. Se calcula el valor de aptitud de ambos individuos (padre y descendiente) y permanece en la población aquel que tiene mejor valor de aptitud. Cada nuevo individuo se genera mediante la expresión: $\bar{X}^{t+1} = \bar{X}^t + N(0, \bar{\sigma})$, donde t es la generación actual y $N(0, \sigma)$ es un vector de números Gaussianos independientes con un valor de media *cero* y desviación estándar σ.

Posteriormente en 1973, Rechenberg [113] propone la estrategia evolutiva $(\mu + 1) - EE$ con μ padres, donde se introduce el concepto de población. En esta versión, se trabaja con varios padres y un sólo hijo que puede reemplazar a uno de los μ padres, aquel que tenga peor valor de aptitud. Además, se establece la relación entre el número de mutaciones realizadas con éxito y el número total de mutaciones en $1/5$, denominándose "regla del éxito $1/5$". Si el resultado obtenido es mayor se incrementa la desviación estándar, en caso contrario se decrementa.

Schwefel [114, 115] introduce la generación de múltiples hijos con $(\mu+\lambda)-EEs$ y $(\mu, \lambda)-EEs$. En $(\mu+\lambda)-EEs$ los μ individuos que sobreviven son los mejores entre padres e hijos. En $(\mu, \lambda)-EEs$ sólo los μ mejores hijos sobreviven.

Las ES se concentran en el proceso evolutivo a nivel de individuos, a diferencia de la EP que lo hace a nivel de especie. Bajo este supuesto, la operación de cruce sí puede ser implementada. Las ES han sido aplicadas con éxito a problemas de

bioquímica, óptica, diseño en ingeniería [116, 117, 118], entre otros.

3.2.3. Algoritmos genéticos

En 1962, Holland [119] hace la primera referencia a sistemas autoadaptativos, utilizando los modelos evolutivos existentes para hacer evolucionar una población según los principios de la evolución natural (reproducción, competición, selección y mutación). Posteriormente en 1975, fue el propio Holland [120] quien define los principios básicos de los GAs.

Los GAs trabajan con una población de individuos, donde cada individuo representa una solución factible al problema propuesto. Cada individuo posee un valor de aptitud que permite saber lo buena que es la solución. La probabilidad de ser seleccionado para el proceso de reproducción y de transmitir su material genético se incrementa cuanto mayor es su valor de aptitud. Los nuevos individuos obtenidos en el proceso de reproducción reemplazarán a los de la población actual, así se transmitirán las mejores características a través de las sucesivas generaciones. El algoritmo básico de un GA se representa en la Figura 3.3.

```
1. Generar aleatoriamente la población inicial.
2. Calcular el valor de aptitud de cada individuo.
3. Aplicar el operador de selección.
4. Aplicar el operador de cruce
5. Aplicar el operador de mutación.
6. Mezclar padres e hijos para obtener la siguiente generación
7. Finalizar si se cumple la condición de parada, en caso contrario
   volver al punto 2.
```

Figura 3.3: Algoritmo básico de un Algoritmo Genético. La población inicial es generada aleatoriamente. El siguiente paso calcula el valor de aptitud de cada individuo. La creación de la población de descendientes es guiada por los operadores de selección, el primero decide los individuos que se reproducirán y el operador de cruce realiza la recombinación de ambos progenitores. A continuación se aplica el operador de mutación para preservar la diversidad. El proceso evolutivo continúa hasta que se cumpla la condición de parada.

Todo GA necesita de cinco componentes para poder ser aplicado [121]: (I) Forma de representar las soluciones al problema (codificación); (II) Proceso de generación de la población inicial; (III) Función de evaluación (aptitud de los individuos);

(IV) Operadores genéticos a aplicar y (V) Especificación de parámetros de los que depende (tamaño de la población, probabilidad de cruce, probabilidad de mutación, número de generaciones, etc.).

Los parámetros iniciales que se proporcionan a un GA definen su comportamiento y han sido ampliamente estudiados a lo largo de los años. En 1975, De Jong [122] analizó los efectos del tamaño de la población y de los operadores de cruce y mutación en el comportamiento de un conjunto de algoritmos genéticos, usados en la optimización de un conjunto de funciones. Este estudio sirvió para demostrar la eficacia de los GAs en la resolución de problemas de optimización. Muchos otros investigadores siguieron trabajando sobre los GAs, entre ellos Goldberg, quien en 1989 [123] explicó de forma clara y concisa los conceptos teóricos y de aplicación de los GAs, lo cual supuso que muchos científicos e ingenieros de campos muy diversos comenzaran a aplicarlos.

Se aconseja la utilización de GAs en problemas donde no se dispone de una técnica especializada, o se desea una técnica híbrida que ya ha sido aplicada en multitud de problemas de optimización entre los que se pueden citar, por ejemplo, el reconocimiento de patrones y predicción de tiempo [124, 125].

3.2.4. Programación genética

La programación genética surge en 1992 de la mano de John Koza [126] y tiene como finalidad hacer evolucionar programas de ordenador de forma automática, dentro de su espacio de búsqueda. Los programas se representan en forma de árbol y trabajan sobre una población finita de soluciones candidatas al problema dado. El algoritmo, al igual que sucede en GA, es dirigido por los operadores genéticos (selección, cruce y mutación), como operadores de búsqueda, y por una función de coste (*fitness*). La Figura 3.4 presenta el algoritmo básico de programación genética. El operador de selección permite seleccionar un individuo para el proceso de reproducción, aunque sólo realiza una réplica del individuo seleccionado. El operador de cruce selecciona 2 individuos, de cada uno de ellos se obtiene aleatoriamente

un nodo e intercambia los subárboles que cuelgan a partir de él. La función que realiza el operador de mutación dentro de la Programación Genética es la de seleccionar un nodo del árbol padre, corta el subárbol que cuelga de él y a continuación lo reemplaza por el árbol generado.

```
1. Generar población inicial.
2. Evaluar cada programa/árbol para calcular el valor de aptitud de
    cada uno.
3. Seleccionar individuos según su valor de aptitud, para el
   proceso de reproducción.
4. Aplicar el operador de cruce
5. Aplicar el operador de mutación.
6. Añadir/eliminar individuos a la población de la nueva generación
   .
7. Finalizar si se cumple la condición de convergencia, en caso
   contrario volver al punto 2.
```

Figura 3.4: Algoritmo básico de Programación Genética. Se genera la población inicial, generalmente de forma aleatoria. El siguiente paso calcula el valor de aptitud de cada programa, representado en forma de árbol. A continuación, el operador de selección elige los progenitores para la reproducción y el operador de cruce realiza la recombinación para crear a los descendientes. El proceso se completa mediante el operador de mutación y el proceso continúa hasta que se cumpla la condición de convergencia.

El campo de aplicación de GP es muy amplio y no está limitado a programas de ordenador, cualquier problema que pueda ser representado mediante un esquema de árbol es proclive a ser optimizado mediante GP.

Sin embargo, GP se encuentra con algunas restricciones que intenta solventar mediante diferentes técnicas: todas las funciones deben tener la propiedad de cierre o estructuras restrictivas que creen individuos correctos (programación genética fuertemente tipada); el lenguaje y los tipos de datos han de ser sencillos; se debe establecer un tamaño máximo para evitar un fuerte incremento en el tamaño de los árboles y, por tanto, una mayor complejidad en la resolución del problema; el operador de cruce puede generar individuos nuevos demasiado parecidos a los progenitores, para evitarlo se generan nuevas versiones del operador.

GP tiene un campo de aplicación muy amplio. De esta forma, se han abordado problemas en el campo de los circuitos electrónicos, resolución de expresiones

matemáticas, diseño de autómatas celulares, etc. Entre los muchos existentes, los trabajos [127, 128, 129], por ejemplo, aplican GP al diseño de circuitos digitales, estrategias de juego o en problemas de regresión simbólica, respectivamente.

3.2.5. Gramáticas evolutivas

Las Gramáticas Evolutivas tienen como base la Programación Genética y resuelven algunos de los problemas identificados anteriormente, como la generación de individuos incorrectos. GE permite la evolución de programas en cualquier lenguaje, para lo cual se apoya en el concepto de representación formal que se utiliza para definir una gramática, dotando a la técnica de flexibilidad y modularidad. En GE la gramática sirve para generar el fenotipo a partir del genotipo. Las reglas de producción de la gramática suelen utilizar la notación *Backus Naus Form* (BNF). Una gramática BNF es una gramática libre de contexto que viene definida por la tupla N, T, P, S donde N es el conjunto de símbolos no terminales, T representa el conjunto de terminales, P es un conjunto de reglas de producción que mapea los elementos de N a T, y S, perteneciente a N, es el símbolo de inicio. La existencia de varias reglas de producción para un mismo elemento de N aparecen separadas por el símbolo ”|”.

El proceso de evolución mediante GEs conlleva la generación de cadenas binarias de longitud variable, sobre las que habitualmente se realizan agrupaciones de 8 bits, cada una de estas agrupaciones se denomina *codón*. El conjunto de valores que representan los codones conforma el genotipo. El genotipo se decodifica mediante el proceso de mapeo para generar el fenotipo del individuo. Así, a partir del símbolo inicial, a cada valor entero del genotipo se le aplica la función módulo con el número de reglas que tiene el símbolo perteneciente a N a decodificar, el resultado se corresponde con una regla de producción de dicho símbolo no terminal en la gramática BNF definida. La utilización de un genoma lineal, y por tanto su similitud con los cromosomas, en lugar de una representación en árbol, favorece la aplicación de los operadores genéticos y permite además, añadir restricciones al

espacio de búsqueda asociado al problema. Todo ello completa el algoritmo que evoluciona hasta obtener una solución al problema, cercana al valor óptimo. En la Figura 3.5 se describe un ejemplo de gramática para la resolución de expresiones matemáticas, y se describe el mecanismo de mapeo para un caso particular, a partir de una cadena de valores enteros dada.

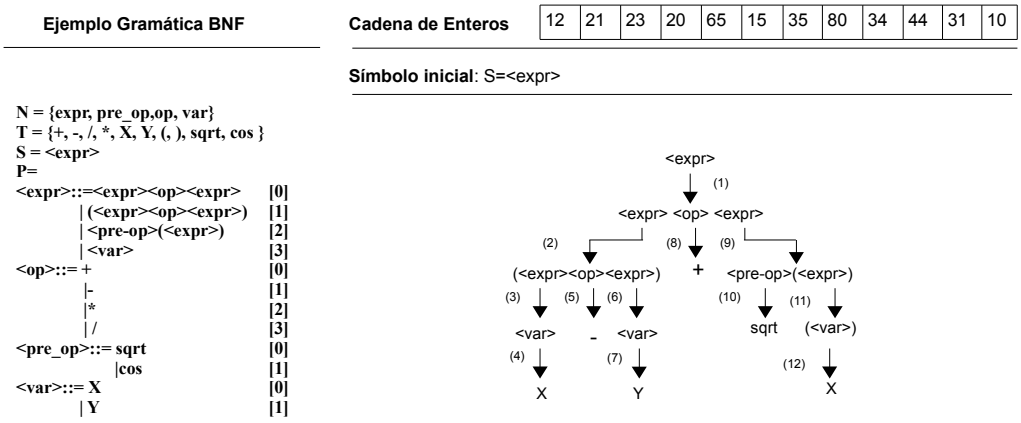


Figura 3.5: Mecanismo de mapeo genotipo a fenotipo para la obtención de un individuo mediante la gramática especificada. El ejemplo muestra una gramática definida para resolver un conjunto de expresiones matemáticas. A partir de la decodificación del símbolo inicial<expr>, se selecciona el primer gen del genotipo, 12. La regla de producción <expr> tiene 4 posibles alternativas, dado que ($12\ MOD\ 4 = 0$), se selecciona la regla de producción 0 que corresponde a <expr><op><expr>. Se toma el símbolo no terminal <expr> y el segundo gen con el valor 21, dado que ($21\ MOD\ 4 = 1$) se selecciona la regla de producción <expr><op><expr>. Continúa el proceso con el símbolo no terminal <expr> y el gen con valor 23, tras la operación ($23\ MOD\ 4 = 3$) la regla de producción a seleccionar es <var>. A continuación se toma el símbolo no terminal <var> y el valor 20 correspondiente al gen 4 y tras la operación módulo ($20\ MOD\ 2 = 0$) se reemplaza por el símbolo terminal X. El proceso continúa de igual forma hasta completar el fenotipo.

A diferencia de la programación genética estándar, que utiliza representación en forma de árbol, el mapeo entre el genotipo y el fenotipo se realiza mediante cadenas binarias de longitud variable, como se ha mencionado anteriormente. El fenotipo es una estructura (programa, expresión, árbol, etc), que se obtiene a través del genoma representado en una cadena binaria. El cromosoma lo componen valores enteros que, en la decodificación, se usan para aplicar las reglas de la gramática, dando lugar al fenotipo. La Figura 3.6 representa los diferentes pasos que sigue el proceso

de GEs [130]: (i) Definición del problema, donde a partir de su particularidad se deberá detallar entre otros, la función de aptitud. (ii) Especificación de la gramática, a partir de la cual se obtendrán las soluciones candidatas al problema dado. (iii) Algoritmo o motor de búsqueda para la obtención de individuos sintácticamente correctos. (iv) Aplicación del algoritmo para evaluar las soluciones candidatas.

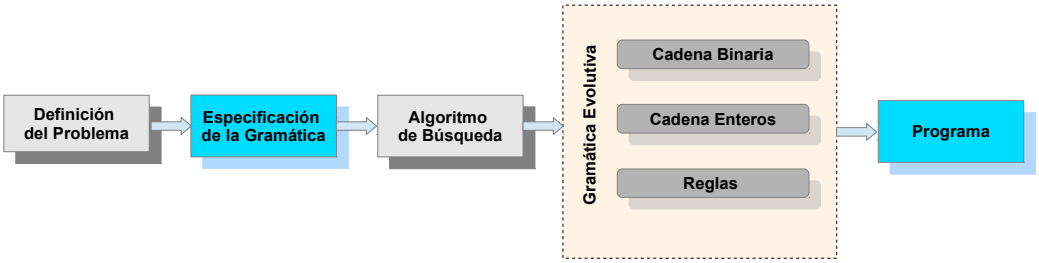


Figura 3.6: Esquema general seguido por el proceso de Evolución Gramatical. En la primera fase se establece la definición del problema, lo cual conlleva la definición de la función de aptitud para la evaluación de las soluciones candidatas. Posteriormente, en base a la fase anterior, se define la gramática: conjunto de terminales, no terminales y reglas. A continuación se procede a la generación de los individuos que conforman la población a evaluar utilizando la gramática definida para realizar el proceso de mapeo, tras lo cual las soluciones pueden ser evaluadas.

Problemas como el modelado financiero [131], estrategias de juego [132], entornos de juego dinámicos [133], diseño de plataformas de juego 2D [134], 3D design [135], entre otros, han sido desarrolladas con éxito mediante GE.

3.2.6. Evolución diferencial

La Evolución Diferencial fue presentada por primera vez por Rainer Storn y KennethStorn Price [106], como una técnica de optimización aplicable a una amplia variedad de problemas cuyo espacio de búsqueda es continuo, y sobre los que trabaja de forma rápida y eficaz. La población está formada por N individuos, donde cada individuo se representa por un vector de valores reales y d dimensiones $x(g)_i$ que se corresponde con una solución al problema dado. Los parámetros del enfoque original son tres: N, F *y* C_r. N, como se ha indicado anteriormente, es el tamaño de la población, el parámetro F se corresponde con la tasa de mutación y C_r es el factor de recombinación. Estos parámetros afectan en gran medida al rendimiento

del algoritmo, por lo cual su estudio ha sido ampliamente abordado. Ajustados los parámetros a los límites impuestos por el problema, se genera aleatoriamente la población inicial de N vectores de d dimensiones de valores reales. Así, x_i^g representa al individuo i-ésimo de la generación g.

$$x_i^g = (x_{i,j}^g, x_{i,j}^g, ..., x_{i,d}^g) \tag{3.1}$$

donde i, g representan el individuo y la generación actual, donde $i = (1, 2, 3, ...N)$, $g = (1, 2, ...M)$.

El proceso de evolución pasa, en primer lugar, por aplicar a todos los vectores los operadores de mutación y cruce y así, se obtiene para cada uno un vector de prueba u_i^g:

$$u_i^g = (u_{i,j}^g, u_{i,j}^g, ..., u_{i,d}^g) \tag{3.2}$$

Para obtener la siguiente generación se aplica el operador de selección entre los vectores x^g y u^g. La aplicación de los operadores de mutación, recombinación y selección continúa hasta alcanzar la condición de parada. Algunas de las estrategias de mutación propuestas han sido: $DE/rand/1$, $DE/best/1$, $DE/rand/2$ y $DE/best/2$ [1]. Para el operador de recombinación las propuestas principales han sido recombinación discreta y aritmética, cada una de ellas con varias alternativas posibles.

El esquema general para DE se presenta en la Figura 3.7, donde se genera aleatoriamente la población inicial P_0 de N vectores, los cuales son evaluados con la función objetivo. Mientras la condición de parada no se cumple, a cada uno de ellos se le aplica mutación para preservar la diversidad y se obtiene v_i^g; el operador de recombinación que permite mantener buenas soluciones de la generación previa y obtener los vectores de prueba u_i^g; el operador de selección que hace que padres x_i^g y descendientes u_i^g compitan para pasar a la siguiente generación.

Desde la aparición de la primera propuesta de DE se han planteado muchas modificaciones, sobre todo incorporando algunos de los nuevos esquemas de mutación

[1] La nomenclatura utilizada es DE/Modo se selección de vectores/Número de vectores. De esta forma, el modo de selección puede ser, entre otros, aleatorio (*rand*), el mejor individuo de la población (*best*). El último valor representa el número de pares de vectores implicados en la operación diferencia (para $DE/rand/1$ $v_i^g = x_{r1}^g + F(x_{r2}^g - x_{r3}^g)$, con selección aleatoria.

```
1. g = 0
2. Generar población inicial P0 de tamaño N, X(0)=(x(0)1, x(0)2
   ,....,x(0)N)
3. Calcular (f(x1^g), f(x2^g), ........,f(xN^g))
5. Mientras (no condición parada)
6. Para cada individuo de la población Pt
7.  v_i^g = mutación(x_i^g)
8.  u_i^g = cruce(x_i^g,v_i^g)
9.  Calcular f(u_i^g)
10.  if f(u_i^g) < f(x_i^g)
11.  then x_i^(g+1)=u_i^g
12.  else x_i^(g+1)=x_i^(g+1)
13. g = g + 1
```

Figura 3.7: Esquema general para DE donde P_t representa la población en la generación t, f es la función objetivo y g la generación en curso. v_i^g representa el vector obtenido tras aplicar el operador de mutación sobre cada individuo de la población y u_i^g es el vector tras aplicar el operador de cruce entre la población de progenitores x_i^g y v_i. Los individuos de x_i^g y u_i^g compiten para formar la población $t + 1$ en siguiente generación.

y recombinación, ya mencionados. En [136] los autores presentan una recopilación de los diferentes enfoques propuestos. Mencionamos, por ejemplo, las propuestas de modificación realizadas por Price en [137], donde se analiza el comportamiento del algoritmo para mejorar el proceso de convergencia y se proponen los valores aconsejables para los parámetros del algoritmo. En 2006 [138] presenta un enfoque auto-adaptativo para el tamaño de la población, el ratio de mutación y cruce. Zhang, presenta otro enfoque adaptativo en [139], donde se propone una nueva estrategia para el operador de mutación denominada *DE/current-to-pbest*, que además permite modificar los parámetros de control de forma adaptativa. En [140] los autores desarrollan un esquema denominado *cDE* (*compact Differential Evolution*) donde aplican dos enfoques de elitismo para mejorar el rendimiento, especialmente para contextos de baja potencia computacional y limitaciones de memoria como, por ejemplo, los microcontroladores y robots comerciales.

3.3. Optimización multi-objetivo

Resolver un problema de optimización es encontrar la solución óptima o un óptimo global dentro del espacio de búsqueda definido. En muchos problemas esta solución óptima debe satisfacer un único objetivo. En este sentido, son muchas las técnicas utilizadas y entre ellas cabe destacar el uso de EAs, que han alcanzando excelentes resultados en una amplia variedad de campos. Sin embargo, no todos los problemas dependen de un único objetivo a satisfacer, la mayor parte de los problemas reales que se plantean para su optimización dependen de varios factores (limitaciones físicas, condiciones del entorno, valores de mediciones etc.), incluso gran parte de estos factores entran en conflicto entre si. Por ejemplo, el diseño de un automóvil depende de muchos factores, algunos de los cuales son el consumo de combustible, velocidad, potencia y ratio de emisiones. Una potencia mayor lleva a alcanzar una velocidad superior, pero incrementa el consumo de combustible y el nivel de emisión de gases perjudiciales para el medio ambiente y la salud de las personas.

En ocasiones algunos problemas multi-objetivo se pueden abordar con técnicas de optimización mono-objetivo, mediante la ponderación de los diferentes objetivos con una única función de coste, siempre y cuando ello sea posible. Sin embargo, el resultado no es el mismo que el obtenido mediante la aplicación de técnicas de optimización multi-objetivo. Algunos de los objetivos pueden alcanzar una mejora muy significativa, mientras que otros apenas mejoren o incluso empeoren y, sin embargo, la ponderación de los objetivos lleve a evaluar positivamente la solución. El Problema de Optimización Multi-objetivo (MOP, *Multi-objective Optimization Problem*) aparece cuando el problema a optimizar debe satisfacer varios objetivos a la vez y no es posible optimizar uno de los valores objetivo sin que alguno de los otros objetivos se vea perjudicado.

En un MOP, se obtiene un conjunto de soluciones debido a los diferentes valores de dominancia de acuerdo a los valores objetivo y que permiten evaluar el grado de adecuación de una solución. Seleccionar la solución a aplicar necesita de un proceso

posterior de toma de decisión por parte de un experto.

Los Algoritmos Evolutivos para Optimización Multi-objetivo son una evolución de los EAs mono-objetivo y surgen ante la necesidad de aportar soluciones a problemas multi-objetivo, hasta entonces, en muchos casos, tratados como mono-objetivo. Aunque la primera implementación aparece en [141], no es hasta finales del siglo XX (década de los 90), cuando el desarrollo de algoritmos multi-objetivo y la comunidad de investigadores en este campo, se incrementa sustancialmente. A continuación se presentan los fundamentos que rigen los algoritmos al abordar un MOP.

La definición de un problema multi-objetivo queda claramente expresada en [142], como: "un vector de variables de decisión que satisfacen un conjunto de restricciones y optimiza una función vectorial cuyos elementos representan las funciones objetivo. Estas funciones conforman una descripción matemática de los objetivos normalmente en conflicto entre sí. Por tanto el hecho de optimizar, significa encontrar una solución que obtenga valores aceptables de las funciones objetivo, para la toma de decisiones."

Más formalmente y sin pérdida de generalidad se puede asumir la siguiente formulación a un problema de minimización con m objetivos:

Minimizar

$$\mathbf{y} = \mathbf{f}(\mathbf{x}) = [f_1(\mathbf{x}), f_2(\mathbf{x}), ..., f_m(\mathbf{x})]$$

Sujeto a

$$\mathbf{x} = (x_1, x_2, \ldots, x_n) \in \mathbf{X}$$

$$\mathbf{y} = (y_1, y_2, \ldots, y_m) \in \mathbf{Y}$$

donde $\mathbf{x}$ es el vector de n variables de decisión, $\mathbf{f}$ es el vector de m funciones objetivo. $\mathbf{X}$ es una región en el espacio de decisión, e $\mathbf{Y}$ es la región posible en el espacio de los objetivos. Una solución $\mathbf{x_1} \in \mathbf{X}$ se dice que domina otra solución $\mathbf{x_2} \in \mathbf{X}$ (denotado como $\mathbf{x_1} \prec \mathbf{x_2}$) si se satisfacen las siguientes dos condiciones,

según el criterio de minimización:

$$\forall i \in 1, 2, ..., m \quad , \quad f_i(\mathbf{x_1}) \leq f_i(\mathbf{x_2})$$

$$\exists j \in 1, 2, ..., m \quad , \quad f_j(\mathbf{x_1}) < f_j(\mathbf{x_2})$$

Si no hay una solución que domina a $\mathbf{x} \in \mathbf{X}$, $\mathbf{x}$ se dice que es una solución óptima de Pareto. En otras palabras, encontrar un óptimo de Pareto significa que no hay otra solución que pueda minimizar algún objetivo sin que, directamente, se incrementen uno o varios de los objetivos restantes. El conjunto de todos los elementos del espacio de búsqueda que no son dominados por algún otro elemento se llama Conjunto Óptimo de Pareto (POS, *Pareto Optimal Set*). La representación del POS, en el espacio objetivo se denomina Frente Óptimo de Pareto (POF, *Pareto Optimal Front*), de un problema multi-objetivo y representa la mejor solución posible con respecto a los objetivos contradictorios. Debido a que varias soluciones pueden ser mapeadas a una misma función multi-objetivo, el POF no contiene necesariamente todos los elementos que están en el conjunto óptimo de Pareto. Un problema de optimización multi-objetivo se resuelve cuando se encuentra su POS. En la práctica, el número de óptimos de Pareto es demasiado grande. La determinación de un único óptimo de Pareto es *NP duro* [143]. De esta forma, el objetivo generalmente es encontrar una aproximación al conjunto de Pareto satisfactoria (normalmente denominado conjunto de Pareto o frente de Pareto en el espacio objetivo), tan cercano como sea posible al conjunto óptimo de Pareto.

La Figura 3.8 ilustra el concepto de dominancia para un problema multi-objetivo con dos funciones objetivo a minimizar $\mathbf{f_1}$ y $\mathbf{f_2}$, donde $\mathbf{f_1}$ está asignada al eje $\mathbf{x}$ y $\mathbf{f_2}$ al eje $\mathbf{y}$. En la gráfica, la línea roja define el POF, el conjunto de soluciones obtenido por el algoritmo se representa por $\mathbf{x_i}$, donde las soluciones en negro son el conjunto de soluciones no dominadas o aproximación al POF y en azul se muestran el conjunto de soluciones dominadas. Se dice que $\mathbf{x_3} \prec \mathbf{x_8}$, para el problema de minimización, porque $\mathbf{f_1}(\mathbf{x_3}) < \mathbf{f_1}(\mathbf{x_8})$ y $\mathbf{f_2}(\mathbf{x_3}) < \mathbf{f_2}(\mathbf{x_8})$. Así, definida una región de interés dentro del espacio de búsqueda, el Frente Óptimo de Pareto (POF) es la representación del conjunto de soluciones no dominadas. Se trata de encontrar el

conjunto de soluciones que más se aproxime al POF (aproximaciones al POF). La implementación del proceso de búsqueda, de forma apropiada, a través del espacio de diseño permite garantizar la diversidad de las soluciones y evitar la convergencia a una solución única. Una medida de la calidad de las soluciones pertenecientes al POF es el nivel de diversidad presente en los puntos incluidos dentro de la región de interés.

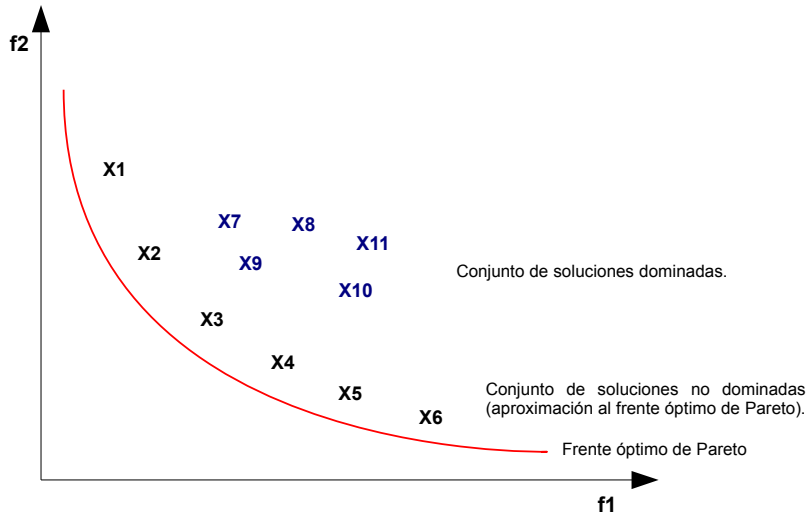


Figura 3.8: La imagen muestra un ejemplo de dominancia de Pareto donde se representa mediante una línea roja el Frente Óptimo de Pareto y el conjunto de soluciones obtenidas por el algoritmo multi-objetivo mediante $\mathbf{x_i}$. Las soluciones más cercanas al POF, en color negro, son el conjunto de soluciones no dominadas o aproximación al frente óptimo de Pareto. Aquellas que aparecen en color azul son el conjunto de soluciones dominadas. Para explicar el concepto de dominancia para un problema de minimización, se seleccionan las soluciones $x_3 \in$ conjunto de soluciones no dominadas y $x_8 \in$ conjunto de soluciones dominadas. Se dice que $\mathbf{x_3} \prec \mathbf{x_8}$ porque $\mathbf{f_1}(\mathbf{x_3}) < \mathbf{f_1}(\mathbf{x_8})$ y $\mathbf{f_2}(\mathbf{x_3}) < \mathbf{f_2}(\mathbf{x_8})$.

Uno de los principales objetivos de un Algoritmo Evolutivo Multi-objetivo (MOEA, *Multi-objective Evolutionary Algorithm*) es preservar la diversidad en la población y evitar la convergencia a una solución única, para lo cual se ha de explorar el espacio de búsqueda de forma adecuada. Algunas técnicas se centran en mejorar la eficiencia del espacio de búsqueda (nichos, *crowding*, etc.) [105]. Sin embargo, a medida que el número de objetivos a satisfacer se incrementa, el foco de atención se centra en soluciones pertenecientes a un subconjunto del espacio de búsqueda, que se denota como Región de Interés (ROI, *Region Of Interest*) [144].

Se identifican dos generaciones de algoritmos multi-objetivo [145], la primera generación se caracteriza por la implementación de algoritmos simples. La segunda generación introduce elitismo mediante el operador de selección $\mu + \lambda$ y una población secundaria. Algunos de los algoritmos más representativos pertenecientes a la primera generación son: Multi-Objective Genetic Algorithm (MOGA) [146], Niched Pareto Genetic Algorithm (NPGA) [147] y Non-dominated Sorting Genetic Algorithm (NSGA) [148]. Con respecto a la segunda generación destacan, entre otros: Strength Pareto Evolutionary Algorithm 2 (SPEA2) [149] y Non-dominated Sorting Genetic Algorithm II (NSGA-II) [150], de este último se habla con más detalle a continuación, por ser el utilizado en esta tesis.

3.3.1. NSGA-II

NSGA-II aparece ante la necesidad de solucionar los problemas detectados en su predecesor NSGA como el alto coste computacional para tamaños de población grandes, la falta de elitismo, dependencia del parámetro σ_{share} y no utilizar dominancia en el proceso de clasificación. Estos problemas lo convirtieron en un algoritmo menos eficiente y propiciaron que otros algoritmos como MOGA y NPGA lo superaran. En la Figura 3.9 se muestra un esquema del algoritmo NSGA-II, donde P_0 es la población inicial y P_t es la población en la generación `t`. Posteriormente la Figura 3.10 muestra su flujo de ejecución [151].

A partir de la población inicial, NSGA-II evalúa los individuos de la población, los clasifica según el rango de no-dominancia asignado y los ordena según cada función objetivo. Los individuos compiten entre ellos como parte natural del proceso de evolución. Los operadores genéticos se aplican a los individuos de la población para crear una población de descendientes.

A continuación, las poblaciones de progenitores y descendientes son combinadas y se aplica elitismo para seleccionar los mejores individuos entre padres e hijos. Los individuos de este conjunto son clasificados por frentes. Aquellos asignados al primer frente representan al vector de soluciones de referencia y se dice que son no

```
1. Generar aleatoriamente población inicial P0 de tamaño N
2. Evaluar individuos P0
3. Clasificar los individuos de P0 según dominancia
4. Aplicación de operadores genéticos (selección, recombinación y mutación) para obtener Q0
5. Para cada generación t
6. Para cada individuo de la población (Padres/Hijos) Pt
7.  Clasificar según la dominancia de Pareto
8.  Generar los vectores de no-dominancia a través del Frente de Pareto
9.  Aplicar crowding y añadir individuos a la siguiente generación
10. Aplicar elitismo
11. Generar siguiente población Pt+1
12. Aplicar operadores genéticos(selección, recombinación y mutación) a Pt+1 para obtener Qt+1
```

Figura 3.9: Esquema general de NSGA-II. Se genera aleatoriamente la población inicial P_0, cuyos individuos se evalúan y a continuación, se clasifican y ordenan de acuerdo al rango de dominancia y cada una de las funciones objetivo. Los operadores genéticos de selección, recombinación y cruce se aplican para obtener la población de descendientes Q_{t+1}. En cada generación las poblaciones de padres e hijos se clasifican en diferentes frentes, utilizando la técnica de nichos y la distancia *crowding*. Los mejores individuos tras aplicar elitismo pasarán a formar parte de población P_{t+1} en la siguiente generación.

dominados. Los individuos que pertenecen al segundo frente son dominados por el primer frente, pero son no dominados si no se tiene en cuenta el primero, y así sucesivamente. El nivel de dominancia permite asignar al individuo un rango, por el cual se ordena y que facilita el paso de los mejores individuos a la siguiente generación. Utiliza la técnica de nichos sobre las variables de decisión y el cálculo de la distancia *crowding* para preservar la diversidad en la siguiente generación y mejorar la exploración del espacio de búsqueda.

Crowding es una técnica que mide la distancia normalizada existente entre una solución y las dos soluciones adyacentes a ella, según los valores que presentan para las funciones objetivo definidas. La Figura 3.11 muestra el esquema para el cálculo de la distancia *crowding* y cómo tomando un punto del frente de Pareto, la disposición de los puntos respecto a este describen un cuboide cuyos vértices representan a las soluciones adyacentes.

La distancia *crowding* permite evaluar la diversidad existente alrededor de una región dada. Los valores más altos de distancia *crowding* indican que las soluciones a las que pertenecen se encuentran en regiones menos pobladas que aquellas soluciones que presentan valores menores en su distancia *crowding*. El cálculo de la distancia *crowding* comienza ordenando descendentemente los individuos de la

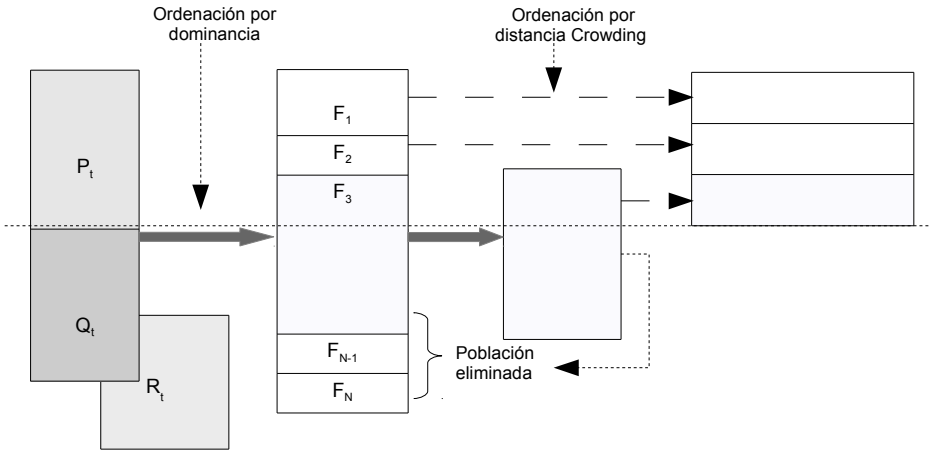
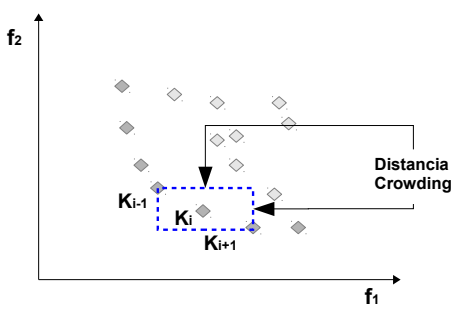


Figura 3.10: Flujo de ejecución de NSGA-II [151]. P_t es la población de padres y Q_t es la población de descendientes en la generación t. Las dos poblaciones se combinan formando la población R_t que se clasifica en frentes F_t y se ordena según el rango de dominancia y cada función objetivo. Los mejores individuos pasarán a la formar parte de población P_{t+1} en la siguiente generación. La técnica de nichos y distancia *crowding* se utiliza para clasificar aquellos individuos que son dominados o no-dominados y así, tomar la decisión de qué individuos pasarán a la siguiente población.

población según el valor obtenido en las funciones objetivo. A los individuos situados en los extremos se les asigna infinito como distancia *crowding*, para el resto de individuos la distancia *crowding* normalizada se calcula como la diferencia entre los valores de las funciones objetivo de las soluciones adyacentes. El valor final de la distancia *crowding* es la suma de todas las distancias de todas las soluciones para cada valor objetivo.

3.3.2. Medidas de calidad

A diferencia de un algoritmo mono-objetivo, cuyo resultado es una solución única, un algoritmo multi-objetivo obtiene como resultado un conjunto de soluciones. Es por esta razón, por lo que se hace necesaria la utilización de indicadores que permitan medir la calidad de las soluciones contenidas en dicho conjunto, con el fin de poder comparar distintos conjuntos de soluciones. A continuación se explican brevemente algunos de los indicadores más utilizados, incluyendo el indicador hipervolumen, utilizado en esta tesis. Se puede obtener información adicional sobre estos y otros indicadores en [105].



Cálculo distancia crowding

Siendo $\mathbf{n}$ *el número de soluciones no dominadas en el conjunto* $\mathbf{S}$
$\mathbf{M}$ = *Objetivos a optimizar*
$\mathbf{I}_m$ = *Conjunto de soluciones ordenado respecto a cada objetivo*
$f_m^{I_j}$ = *Valor de la solución* $\mathbf{j}$ *en relación al objetivo* $\mathbf{m}$
f_m^{max} *y* f_m^{min} = *Valores máximo y mínimo para la función objetivo* $\mathbf{m}$
$\forall\, i \in S\ di = 0$
$\forall\, m \in M$
I_j^m = *ordenar* $\mathbf{S}$ *respecto al objetivo* $\mathbf{m}$
$dI_1^m = \infty$
$dI_n^m = \infty$
for $\mathbf{j} = 2$ to $\mathbf{n}-1$

$$dI_j^m = dI_j^m + \frac{\left(f_m^{I_{j+1}^m} - f_m^{I_{j-1}^m}\right)}{\left(f_m^{max} - f_m^{min}\right)}$$

Figura 3.11: La parte izquierda de la figura muestra la representación del cálculo de la distancia *crowding* sobre un conjunto de soluciones en un frente de Pareto determinado. El cálculo de la distancia *crowding* entre los puntos de cada frente, permite determinar qué regiones del espacio de soluciones están más o menos pobladas. El objetivo es mejorar la diversidad en la población de la siguiente generación. A la derecha se detalla el pseudocódigo de dicho cálculo.

Indicador hipervolumen (I_H) [152]: también conocido como S-metric, es ampliamente utilizado para comparar la calidad de las aproximaciones al frente de Pareto, resultado de la optimización multi-objetivo. Para un problema propiamente multi-objetivo, el hipervolumen representa el espacio de **n** dimensiones cubierto por el conjunto de soluciones de un frente de Pareto conocido respecto a un punto fijo, denominado punto de referencia. Cuando se aborda un problema con dos objetivos, I_H es el área cubierta o la suma de las áreas rectangulares delimitadas por el punto de referencia y las funciones objetivo definidas. La siguiente ecuación, donde Q es un conjunto de aproximación al POF y v_i representa el hipervolumen de la solución $i \in Q$, respecto al punto de referencia establecido, expresa matemáticamente la definición de I_H:

$$I_H(Q) = \bigcup_{1}^{|Q|} v_i \tag{3.3}$$

Para un conjunto de referencia R y un conjunto de aproximación Q_i con $i = 1...N$, se puede expresar I_H como:

$$I_H(Q_i) = |Q_1 - R_1|\ x\ |Q_2 - R_2|\ x.....x\ |Q_M - R_M| \tag{3.4}$$

donde el número de soluciones pertenecientes al frente de Pareto se representa por N y el número de funciones objetivo por M.

Cuando se aborda un problema de minimización las soluciones con menor valor son consideradas las mejores soluciones. La Figura 3.12 muestra cómo el indicador de hipervolumen mide el tamaño de una región dominada, con respecto a un punto de referencia. $I_H A$ e $I_H B$ representan el área cubierta por el conjunto de soluciones de A y B, respectivamente.

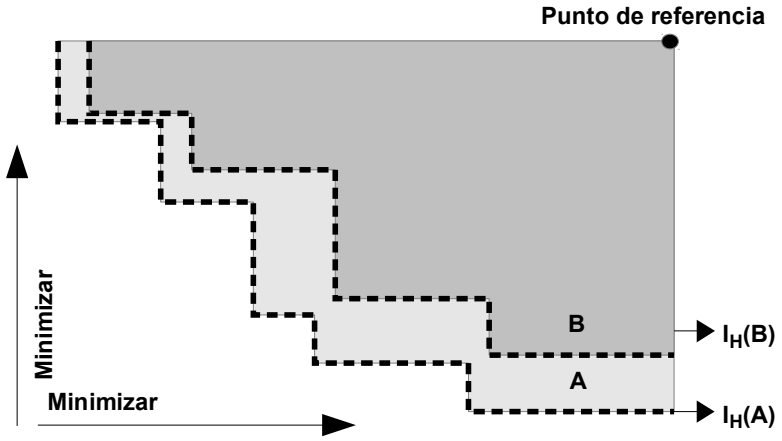


Figura 3.12: Dominancia entre dos regiones utilizando el indicador hipervolumen, de acuerdo a un punto de referencia. En la figura aparecen las regiones A y B, donde $A \triangleleft B$, teniendo en cuenta su proximidad al frente de Pareto.

El indicador hipervolumen tiene importantes cualidades [153], como la comparación de soluciones obtenidas de diferentes algoritmos multi-objetivo, la precisión al medir la proximidad de un conjunto de soluciones al frente de Pareto y su distribución a lo largo de este y la capacidad de indicar que un conjunto de soluciones A no es peor que otro B, bien porque $A \prec B$ o $A \triangleleft B$ [2].

Calcular el indicador hipervolumen tiene un coste computacional alto que aumenta exponencialmente conforme crece el número de objetivos, pero la calidad de las soluciones que se obtienen es muy alta.

Indicador unario epsilon I_ϵ [153]: dispone de la versión aditiva y la multiplicativa. Bajo esta última, la evaluación de dos conjuntos de aproximación (P, Q), $I_\epsilon(P, Q)$, define el mínimo factor ϵ por el cual se ha de multiplicar cada punto per-

[2] $\triangleleft$ o $\preceq$: expresan dominancia débil, al menos existe $x_2 \in B$ que es débilmente dominado por al menos un $x_1 \in A$ siempre que $A \neq B$. $\prec$: operador de dominancia, al menos existe $x_2 \in B$ que es dominado por al menos un $x_1 \in A$.

teneciente a P, a fin de obtener un conjunto de aproximación débilmente dominado por Q.

$$I_{\epsilon(P,Q)} = \underset{\epsilon \in R}{Inf}\{\forall x^2 \in Q \; \exists \, x^1 \in P : x^1 \preceq x^2\} \tag{3.5}$$

Así el indicador multiplicativo epsilon para un frente A se define como:

$$I_\epsilon(A) = I_\epsilon\,(A, R) \tag{3.6}$$

donde R es un conjunto de puntos referencia.

Los indicadores Epsilon e Hipervolumen se basan en principios distintos que pueden llevar a una ordenación diferente de las soluciones para dos conjuntos de aproximación y, por tanto, concluir que ambos conjuntos son incomparables. El coste computacional es menor siendo de orden $O(n \times |A| \times |R|)$, siendo n el número de objetivos.

Indicadores R $\left(I^1_{R2},\, I^1_{R3}\right)$ [154]: permiten evaluar y comparar dos conjuntos de aproximaciones. Para ello se realiza una estimación sobre la cercanía de una solución al frente de Pareto o punto de referencia. Se basa en un conjunto de funciones parametrizadas, que hacen corresponder los vectores de los valores objetivo con valores reales para evaluar los conjuntos de aproximación al frente de Pareto. Las mejores soluciones son aquellas que obtienen los valores más pequeños del indicador $R2$. Formalmente los autores definen los indicadores R como:

$$I_{R2}\,(A, B) = \frac{\sum_{\lambda \in \Lambda}[u*(\lambda, A) - u*(\lambda, B)]}{|\Lambda|} \tag{3.7}$$

$$I_{R3}\,(A, B) = \frac{\sum_{\lambda \in \Lambda}[u^*\,(\lambda, B) - u^*\,(\lambda, A)]/u^*\,(\lambda, B)}{|\Lambda|} \tag{3.8}$$

donde u representa una función de utilidad (por ejemplo una función lineal), Λ es el conjunto de parámetros de las funciones de utilidad, $\lambda = (\lambda_1, ...\lambda_n)$ representa un vector de pesos particular, u^* es el máximo valor obtenido por una función utilidad u_λ con un vector λ sobre un conjunto de aproximación A. El coste computacional de los indicadores R es de orden $O(n \times |\Lambda| \times |A| \times |R|)$, siendo n el número

de objetivos. Junto con el indicador hipervolumen el indicador R obtiene soluciones de calidad y es recomendado por muchos autores.

En esta tesis se han utilizado técnicas metaheurísticas para llevar a cabo procesos de optimización mono-objetivo y multi-objetivo. Se ha aplicado NSGA-II para la optimización del banco de registros y la memoria cache. Otras propuestas de optimización de la memoria cache se han abordado mediante GE y DE. La gestión de memoria dinámica ha sido dirigida por un algoritmo multi-objetivo basado en GE. En los siguientes capítulos se detallan los procesos de evaluación y optimización realizados sobre los niveles de jerarquía de memoria: banco de registros, memoria cache y gestión de memoria dinámica.

Capítulo 4

Gestión térmica del banco de registros

4.1. Introducción

Tal y como se ha mencionado en el Capítulo 2, los avances en la tecnología de integración de circuitos han provocado que el consumo y la densidad de potencia de los microprocesadores aumente. Incluso utilizando niveles bajos de voltaje, el continuo incremento de la densidad del chip provoca un exceso de calor que hay que disipar. El problema térmico se ha convertido en uno de los temas más abordados no sólo por la influencia que tiene sobre la vida de las baterías, sino también sobre la fiabilidad de los sistemas. Además, los sistemas de refrigeración necesarios para evitar daños en los componentes electrónicos incrementan notablemente su coste. En este contexto, uno de los principales objetivos durante el diseño de circuitos es el mantenimiento de la temperatura dentro de unos niveles que garanticen el rendimiento del sistema y evite el efecto dañino que sobre ellos produce [13, 44].

El banco de registros, que soporta un elevado número de accesos, se ha identificado, por ejemplo en [155], como un componente que alcanza altas temperaturas, incluso como uno de los puntos más calientes en un sistema empotrado [46], lo cual conlleva un alto consumo de energía. Además, la necesidad de incrementar el paralelismo para mejorar el rendimiento, ha llevado a incrementar el número de registros y el número de puertos. En consecuencia, el número de accesos también se ha incrementado, lo que aumenta la degradación y puede aumentar la temperatura.

El incremento de temperatura incide negativamente sobre el rendimiento, el consumo de potencia y afecta a la fiabilidad y al coste. Muchos autores, algunos de los cuales han sido mencionados en el Capítulo 2, han propuesto técnicas que permiten minimizar el problema térmico. Estas decisiones, las cuales afectan a la estructura, configuración y comportamiento del procesador deben ser tomadas en las etapas iniciales de diseño [156]. Sin embargo, el estudio térmico requiere un largo tiempo de simulación y depende en gran medida de los materiales y las técnicas de ensamblado y construcción [157], así pues su estudio durante las etapas iniciales no es una tarea simple. Todo ello ha llevado a proponer diferentes técnicas de simulación que facilitan el estudio de la evolución de la temperatura en los componentes del procesador.

Este capítulo estudia el problema térmico en el banco de registros y se ha seleccionado para ello una estructura basada en las arquitecturas VLIW y ARM. El motivo de elegir VLIW y ARM se basa en el hecho de que ambas arquitecturas tienen un comportamiento opuesto. En el caso de VLIW el acceso es esencialmente paralelo, en una sola instrucción se empaquetan múltiples operaciones independientes y se dispone de múltiples *pipelines*. Mientras que en el caso del ARM, una instrucción es una operación simple y de esta forma el acceso a un mismo subconjunto de registros se realiza de forma repetitiva. Esto nos permite analizar el impacto térmico en lo que podemos suponer son los dos polos opuestos en cuanto al estrés generado sobre los registros. La solución propuesta, para abordar la reducción de temperatura, se basa en la aplicación de Algoritmos Evolutivos, más concretamente MOEAs. El motivo de elegir MOEAs es doble: por un lado permite la minimización de varios objetivos de forma extremadamente sencilla e imparcial. Por otro lado, la paralelización mediante modelos de grano grueso como el uso de islas es también prácticamente automática (en caso de ser necesario). Por supuesto que otras técnicas son igualmente válidas, pero el estudio de técnicas de optimización no es el objetivo de este capítulo.

En consecuencia, el objetivo principal de este estudio se basa en el análisis del

incremento de temperatura debido al acceso al banco de registros y su influencia sobre la temperatura del microprocesador. La primera fase consistirá en calcular el impacto térmico durante la ejecución de un conjunto de aplicaciones. Una vez finalizada esta fase se propone la aplicación de un MOEA para estudiar la posibilidad de reducir ese incremento de temperatura mediante una redistribución espacial de los accesos al banco de registros. Para llevar esto a cabo, en primer lugar hay que determinar cuál es la disposición geométrica de los registros y, a continuación, determinar la posición de cada registro en esta configuración de acuerdo a la densidad de potencia[1] alcanzada. Este enfoque permite alejar[2] entre sí aquellos registros que soportan un número mayor de accesos, permitiendo reducir el incremento de temperatura debido a la transferencia de calor.

Para medir el comportamiento térmico, se ha seleccionado un conjunto de aplicaciones multimedia y un esquema basado en simulación. El objetivo es explorar las diferentes alternativas en cuanto al diseño del banco de registros, analizar su comportamiento térmico y buscar una configuración óptima. En la siguiente sección se detalla el modelo térmico seleccionado para abordar esta parte de la tesis.

4.2. Modelo térmico

La transferencia de calor en un sólido o conductividad térmica se produce por un diferencial de temperatura y la transferencia de calor tiene lugar por la vibración de los átomos y moléculas de un sólido, ante un incremento de energía y hacia aquellos átomos y moléculas que presentan una menor energía [158].

Para estudiar el problema térmico asociado a los circuitos integrados es necesario comprender cómo se realiza el proceso de transferencia de calor entre fuentes y

[1] La densidad de potencia se define como la potencia por unidad de área, generalmente, mW/cm^2. En un circuito electrónico al incrementar el número de componentes, aumenta el consumo de potencia y por tanto, la necesidad de disiparla para que no afecte a la fiabilidad del sistema.

[2] El "alejamiento" de los registros significa asignar registros lógicos con una frecuencia alta de accessos, a registros que se encuentran separados físicamente en el banco de registros. Esta asignación se puede realizar en el proceso de compilación en lugar de utilizar la asignación inicial antes de aplicar ninguna técnica de optimización. Durante la compilación la fase de asignación de registros aplica técnicas de optimización para distribuir los accesos a lo largo del banco de registros evitando que todos ellos coincidan en un área reducida y se incremente su temperatura.

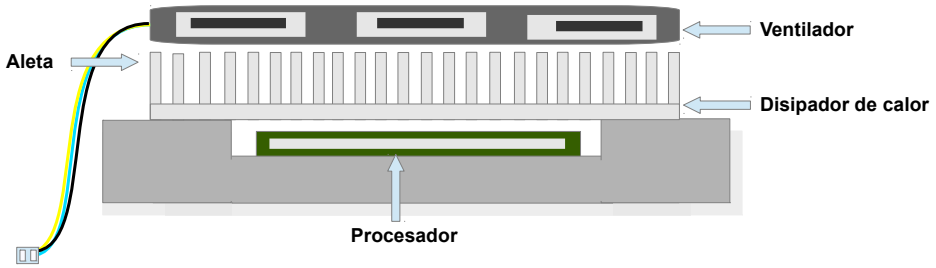


Figura 4.1: Componentes implicados en la disipación de calor de un circuito electrónico, como es un microprocesador. Elementos como disipadores de calor y aletas permiten incrementar la disipación de calor por convección y evacuar el calor al exterior con la ayuda del ventilador.

sumideros, hasta evacuar el calor al exterior. Las fuentes de calor son los transistores y las interconexiones; los sumideros de calor son los elementos térmicamente pasivos. La Figura 4.1 muestra los componentes implicados en la disipación de calor generada en un microprocesador, como ejemplo de circuito electrónico. La conducción del calor se rige por la Ley de Fourier que dice que la transferencia de calor entre dos puntos situados a una determinada distancia es proporcional al gradiente de temperatura[3] y al área que atraviesa. Para un espacio unidimensional se tiene que :

$$Q = -KA\frac{\triangle T}{\triangle Y} = -KA\frac{\partial T}{\partial Y} \tag{4.1}$$

donde Q es el flujo de calor, A el área normal en la dirección del flujo de calor, $\frac{\partial T}{\partial Y}$ es el incremento de temperatura en la dirección Y y K es la conductividad térmica del material o constante de proporcionalidad.

La Figura 4.2 muestra el gradiente de temperatura que se produce entre dos puntos x e y (con una resistencia térmica R_k y separados por una distancia L), y la dirección del flujo de acuerdo al criterio que establece que el flujo de calor se dirige de un punto con temperatura mayor a otro con temperatura menor.

El modelo utilizado para el análisis térmico sigue el enfoque presentado en [159], donde se puede encontrar información más detallada. En dicho modelo se considera que el problema térmico en un circuito integrado (IC, *Integrated Circuit*) puede ser

[3]Magnitud vectorial que mide la variación de la temperatura por unidad de distancia.

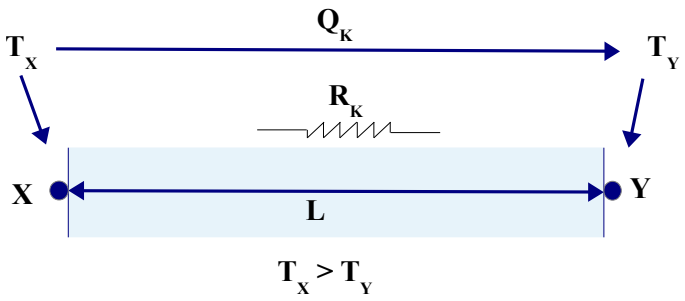


Figura 4.2: Flujo de calor que se establece entre dos puntos x e y en la dirección del eje x, donde se produce un gradiente de temperatura dado por $T_x > T_y$, siendo T_x la temperatura en el punto x y T_y la temperatura en el punto y. Q_k es la potencia calorífica y R_k la resistencia térmica.

abordado mediante ecuaciones diferenciales. Para llevarlo a cabo, un IC se subdivide en elementos volumétricos, cuya distribución térmica puede ser calculada de forma independiente y mediante una ecuación diferencial, la cual depende del tiempo, del material, de la disipación de potencia y de la temperatura de los elementos vecinos. A continuación se muestran las ecuaciones que rigen la conducción térmica en los circuitos integrados, y en consecuencia, el flujo de calor, cuya magnitud es vectorial y que puede ser expresado en forma diferencial:

$$\rho c \frac{\partial T(\vec{r},t)}{\partial t} = \nabla \cdot \left[K(\vec{r}) \nabla T(\vec{r}.t) + p(\vec{r},t) \right] \tag{4.2}$$

Donde ρ representa la densidad del material, c es la capacidad calorífica de la masa, $T(\vec{r},t)$ y $K(\vec{r})$ son la temperatura y la conductividad térmica del material en la posición $\vec{r}$ y en el tiempo t, $p(\vec{r},t)$ es la densidad de potencia de la fuente de calor. Sin embargo, dado que el objetivo es conocer la distribución de la temperatura en un sólido, se hace imprescindible tener en cuenta las condiciones físicas de ese sólido en las fronteras. Conocida la distribución de temperatura, el flujo de calor puede ser obtenido aplicando la ecuación de Fourier. Asi, la Ecuación (4.2) con la aplicación de las condiciones de frontera queda como:

$$K(\vec{r},t) \frac{\partial T(\vec{r},t)}{\partial n_i} + h_i T(\vec{r},t) = f_i(\vec{r},t) \tag{4.3}$$

Para la Ecuación (4.3), n_i es la dirección normal hacia el contorno de la superficie i, h_i es el coeficiente de transferencia del calor; y f_i es una función arbitraria a

la superficie i.

Debido a que las propiedades termofísicas del material dependen de la temperatura, el problema térmico se rige por una ecuación diferencial no lineal. La conductividad térmica del material depende de la temperatura y de la capacidad que tiene un material para transmitir calor, y viene definida por la función: $K = K_0 \left(\frac{T}{300}\right)^{-\eta}$ donde k_0 es el valor de la conductividad del material a $300K$ y η es una constante para el material específico. Sin embargo, para este enfoque se ha utilizado la aproximación de la conductividad térmica a una constante, que en el caso del silicio se corresponde con 148 $\frac{W}{m \cdot K}$.

La resolución de la ecuación diferencial se realiza mediante la aplicación de métodos numéricos, basada en la discretización del medio. En esta tesis, se utiliza el método numérico de las diferencias finitas (particularización de los métodos de elementos finitos de análisis multidimensional). Para el proceso de discretización, el circuito integrado se divide en elementos térmicos discretos que interactúan entre sí mediante la difusión de calor. Cada elemento tiene una disipación de potencia, temperatura, capacitancia y resistencia térmica al elemento adyacente. En un punto interno de un circuito electrónico representado por un conjunto de celdas (*grid*), la ecuación en forma discreta que define su comportamiento térmico es:

$$\rho c V \frac{T_{i,j}^{m+1} - T_{i,j}^{m}}{\triangle t} = -2\left(G_x + G_y\right) T_{i,j}^{m} + G_x T_{i-1,j}^{m} + G_x T_{i+1,j}^{m} + G_y T_{i,j-1}^{m} + G_y T_{i,j+1}^{m} + V_{pi,j} \quad (4.4)$$

donde i, j son los desplazamientos a través de los ejes x e y; $\triangle x$ e $\triangle y$ los pasos discretos en los ejes x e y; $V = \triangle x \triangle y$. G_x y G_y se corresponden con las conductividades térmicas entre elementos adyacentes; $G_x = K\frac{\triangle y}{\triangle x}$ y $G_y = K\frac{\triangle x}{\triangle y}$ y $\triangle t$ es el paso discreto de tiempo t. En el caso que nos ocupa $\triangle x = \triangle y$, por tanto, $G_x = G_y = k$.

Una vez realizada la discretización, el análisis térmico puede ser resuelto utilizando la siguiente ecuación:

$$\frac{C dT(t)}{dt} + AT(t) = Pu(t) \quad (4.5)$$

donde C es la matriz de capacitancia térmica de NxN, A es la matriz de conductividad térmica representada como una matriz dispersa de NxN. $T(t)$ y P son los vectores de temperatura y potencia de $Nx1$ y $u(t)$ la función de paso de tiempo. Dado que se pueden llegar a manejar matrices de gran tamaño, se hace fundamental que las operaciones a realizar por el método numérico se realicen con precisión y rapidez.

Presentadas las ecuaciones según [159], hay que decidir el tipo de análisis a realizar. Para la transferencia de calor hay dos tipos: análisis térmico en estado transitorio, donde se tiene en cuenta la variación respecto al tiempo; y análisis térmico en estado estable o estacionario, donde no se tiene en cuenta la variación.

El análisis térmico en estado transitorio se caracteriza por saber la temperatura en función del tiempo y además de propiedades como la conductividad térmica, la densidad y el calor específico se han de especificar la temperatura inicial, el incremento de tiempo y la curva de potencia. El tiempo se divide en intervalos de pequeño tamaño para estimar con precisión el comportamiento térmico del IC. En cada intervalo se analiza la temperatura con respecto a la temperatura inicial.

El análisis térmico en estado estacionario se caracteriza porque el estudio se realiza cuando se ha alcanzado el estado de equilibrio. Es decir, el flujo de calor y el consumo de potencia no cambian en función del tiempo. La conductividad térmica y el consumo de potencia en el caso peor son las propiedades esenciales que se necesitan en este tipo de análisis. Así, los términos que dependen del tiempo desaparecen y la Ecuación (4.5) se convierte en la Ecuación (4.6), que permite calcular la temperatura de un circuito electrónico representado mediante un conjunto de celdas, en base a la potencia de cada registro y su conductividad térmica:

$$AT = P \rightarrow T = A^{-1}P \tag{4.6}$$

donde la temperatura T se calcula como la inversa de la matriz de conductividad térmica A^{-1} por el vector de potencias P.

Debido a que la matriz A se incrementa de forma cuadrática en función del número de elementos del grid o celdas que representan el circuito electrónico, la

solución directa no es posible. Los métodos numéricos iterativos ofrecen solución a estos problemas de gran tamaño y su calidad viene definida por la tasa de convergencia. La tasa de convergencia está directamente relacionada con la capacidad para eliminar los errores de baja frecuencia y limitan el rendimiento de los métodos iterativos clásicos como Jacoby y Gauss-Siedel. Las métodos *multigrid* [160] son un conjunto de algoritmos para la resolución de ecuaciones diferenciales que utilizan discretización jerarquizada. Utilizan la extrapolación de mallas de grano fino a grueso y están considerados como uno de los métodos más rápidos. Estos métodos reducen los errores de baja frecuencia y por tanto, mejoran la tasa de convergencia y en consecuencia, el rendimiento y la eficiencia.

En esta tesis se utiliza el análisis térmico en estado estacionario y el método multigrid, basado en el estudio realizado en [159], donde los métodos multigrid se utilizan para realizar el análisis térmico en estado estable del circuito integrado, para calcular el incremento de temperatura que producen los accesos al banco de registros del microprocesador. El enfoque que se presenta utiliza estructuras identificadas dentro de las arquitecturas VLIW y ARM, arquitecturas ya justificadas en la introducción. Con ese objetivo en mente, el área asociada al banco de registros se divide en celdas unitarias de igual tamaño. A efectos de cálculo, el banco de registros se representa como una matriz de NxM celdas, donde a cada registro le corresponden un conjunto de celdas y para cada una de ellas se calcula la potencia consumida debida al número de accesos al registro al cual pertenece. El número de accesos, en esta tesis, se obtiene a través de un perfil del comportamiento del conjunto de aplicaciones multimedia seleccionadas.

4.3. Arquitecturas analizadas

El banco de registros de las arquitecturas VLIW y ARM es el objetivo del estudio de este capítulo de la tesis. Siendo conscientes de las múltiples variantes, se ha elegido para cada arquitectura una configuración típica. La primera arquitectura dispone de un banco de registros con 32 registros de propósito general y la segun-

da dispone de 37 registros, 20 de ellos sólo están disponibles cuando el procesador está en un determinado modo de trabajo, 16 se encuentran disponibles para todos los modos de trabajo excepto para el modo supervisor que tiene un subconjunto de ellos y es en estos en los que nos centramos. Las dos arquitecturas presentan un patrón de comportamiento diferente que nos permite analizar el impacto térmico debido a la presión por el gran número de accesos al banco de registros, durante la ejecución de las aplicaciones. VLIW basa su comportamiento en la detección de paralelismo y la replicación de unidades funcionales. ARM posee un subconjunto de registros que replica para cada uno de los modos del procesador y que soportan los accesos continuos.

En ambas arquitecturas, el compilador tiene asignadas las tareas de minimizar las paradas debidas a los riesgos de datos y reordenar las instrucciones, evitando que sea el hardware el que tenga que chequear las dependencias, función que también pasa a formar parte del compilador. De igual forma, la incorporación al compilador de técnicas de optimización para la asignación de registros permite alejar aquellos registros que soportan mayor número de accesos y por consiguiente, reducir el impacto térmico. En consecuencia, el hardware se simplifica (reducción de la lógica de control, incremento de la frecuencia de reloj, espacio para nuevas unidades funcionales, etc.), y se mejora el rendimiento. A cambio se requieren otros ajustes que llevan al diseño, por ejemplo de las dos arquitecturas seleccionadas en esta tesis.

4.3.1. Arquitectura VLIW

La arquitectura *VLIW (Very Long Instruction Word)* surge como alternativa a los procesadores superescalares y el compilador se convierte en una pieza fundamental en el proceso de optimización [10]. El compilador de VLIW, tal y como muestra la Figura 4.3, además de las tareas mencionadas anteriormente debe realizar el desenrollado de bucles, la planificación de las instrucciones y la asignación eficiente de los recursos hardware, como son los registros.

La rigidez que presentaban las primeras versiones de VLIW, respecto al for-

mato de instrucciones, hacía necesario recompilar los programas para adecuarlos a las exigencias de cada nueva versión del hardware. Posteriormente, se introdujeron mejoras e innovaciones que flexibilizaron estos aspectos y mejoraban el rendimiento, aunque sigue siendo el compilador el encargado de realizar la mayor parte del trabajo y planificar las instrucciones para incrementar el paralelismo.

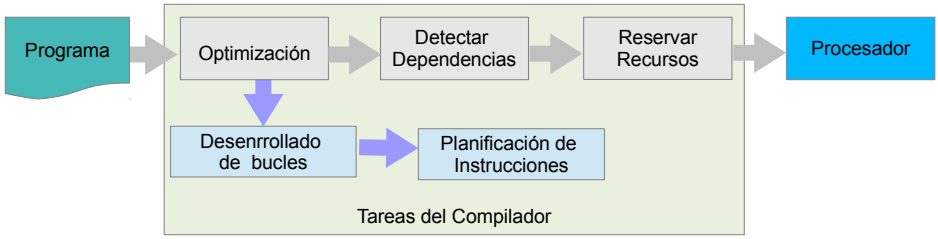


Figura 4.3: Flujo de ejecución de un programa sobre una arquitectura VLIW durante el proceso de compilación. La fase de optimización se encarga del desenrollado de bucles y la planificación de las instrucciones. Posteriormente, el compilador se encarga de la detección de las dependencias de datos y de la asignación de recursos físicos.

VLIW permite la emisión de múltiples e independientes instrucciones hacia las unidades funcionales. Para mantener ocupadas a las unidades funcionales, el código debe proporcionar suficiente paralelismo. En esta dirección, técnicas como el desenrollado de bucles y la planificación de instrucciones, dentro del propio cuerpo de bucle, ayudan a que los *slots*[4] estén ocupados y aumente el número de operaciones a realizar en paralelo. A veces, también se hace necesario la aplicación de técnicas de planificación local[5].

De esta forma, VLIW se posiciona mejor cuando se incrementa el ratio de emisión, lo cual permite aprovechar el paralelismo a nivel de instrucción simplificando el hardware y reduciendo la potencia y el consumo. Sin embargo, debido a que el peso de las optimizaciones las realiza el compilador, este se hace más complejo y su calidad incide directamente en el rendimiento del procesador. Además, también posee ciertas limitaciones, como la dependencia de las versiones de hardware, el

[4]Número de operaciones que se pueden realizar simultáneamente por instrucción

[5]Técnicas que permiten la optimización de bloques básicos de código: *folding* (reemplazar soluciones por su resultado), propagación de constantes (una variable se considera una constante en el intervalo de dos asignaciones), reducción de potencia (simplificación de operaciones) y reducción de subexpresiones comunes (calcular una sóla vez expresiones recurrentes)

desperdicio de memoria en instrucciones NOP[6] y el conflicto entre unidades funcionales al acceder al banco de registros, entre otros. Muchos de estos problemas se solventaron con la aparición del enfoque *EPIC* que se aplicó en el diseño de la arquitectura IA-64, donde el procesador *Itanium*^{TM} fue la primera implementación. Por otra parte, VLIW se ha situado muy bien en sistemas empotrados para el tratamiento de señales digitales (*DSP*) y procesamiento multimedia, donde la compañía *Texas Instruments* [161] ha desarrollado procesadores con muy buenos resultados.

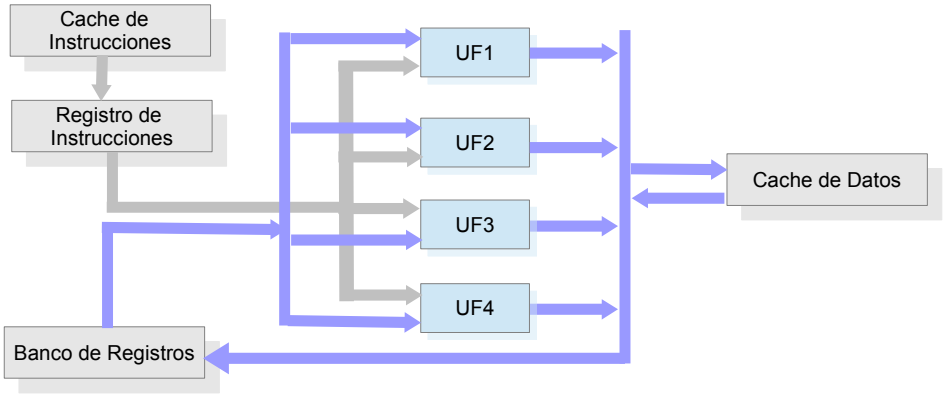


Figura 4.4: Esquema general de una arquitectura VLIW básica. Se compone de 4 unidades funcionales que permiten ejecutar varias operaciones de forma simultánea y así, mantenerlas ocupadas e incrementar el rendimiento.

La arquitectura VLIW, cuyo esquema general se puede observar en la Figura 4.4, proporciona un conjunto de instrucciones similares a las instrucciones RISC, que poseen un conjunto de registros de propósito general y que operan sobre la memoria, es decir, leen un dato de memoria y lo llevan a un registro (*load register from memory*), o almacenan el contenido de un registro en memoria (*store register to memory*). Sin embargo, la longitud de las instrucciones VLIW es mayor para permitir varias operaciones independientes (128 - 1024 bits). El objetivo fundamental es mantener las unidades funcionales ocupadas y para ello el banco de registros juega un papel fundamental. Los operandos se leen del banco de registros y se llevan a las unidades funcionales para su procesamiento, tras lo cual el resultado se lleva de nuevo al banco de registros, desde donde puede ser llevado a memoria o realizar un

[6]NOP es un tipo de instrucción que no realiza trabajo alguno y su propósito es producir un pequeño retardo que ayuda, por ejemplo, a prevenir riesgos.

nuevo procesamiento.

4.3.2. Arquitectura ARM

La fabricación de sistemas empotrados se ha incrementado considerablemente durante la última década. Estudios como el realizado por IDC *The Next Big Opportunity* en el año 2011 [162], predecían que para el año 2015 el número de unidades vendidas de sistemas empotrados rondaría los 11500 millones. Algunos de los sectores que llevarían a alcanzar este número son los de automoción, industrial, aparatos electrónicos, telefonía móvil, comunicaciones, sistemas biométricos y equipamiento médico. Estas cifras han hecho que Intel entre con fuerza en la lucha por este mercado. Sin embargo, la arquitectura ARM ocupa un puesto predominante gracias, por una parte al soporte proporcionado por Google, Apple y Microsoft [163]. ARM comienza su andadura en 1983 de la mano de la empresa ACORN y el primer prototipo se denominó ARM1, pero la comercialización no se inició hasta la versión ARM2. A finales de la década de los 80 se unió a Apple y en 1990 se creó la empresa Advanced RISC Machine que sería la encargada de los siguientes desarrollos de ARM. Con la versión ARM7TDMI, gracias a su bajo consumo, se obtuvo un gran éxito que sirvió para obtener una posición predominante en la implementación de dispositivos móviles.

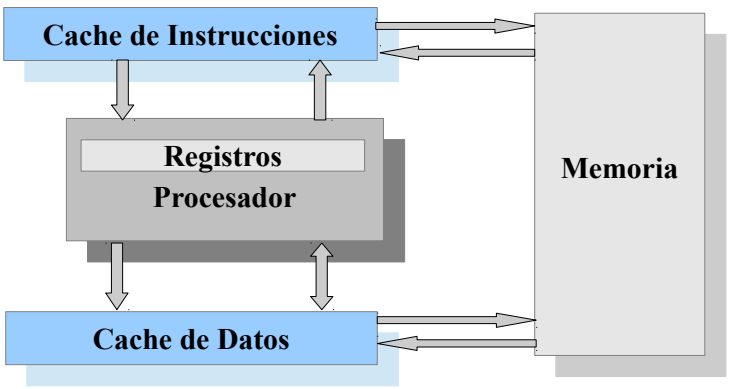


Figura 4.5: Organización básica para una arquitectura ARM según el modelo Harvard. Se muestra el procesador y el sistema de memoria que lo componen los registros, el subsistema de memoria cache separada para instrucciones y datos y la memoria principal.

La arquitectura ARM está basada en la arquitectura RISC, que necesita que los datos estén en los registros del banco de registros para realizar su procesamiento y así, evitar la realización de operaciones directamente sobre la memoria porque son más lentas. RISC reduce la complejidad de las instrucciones e incrementa el aprovechamiento del *pipeline*[7] mediante la organización de las instrucciones y la reducción del número de ciclos por instrucción. Se deja en manos del compilador las tareas de optimización y planificación de las instrucciones para la ejecución en paralelo [10]. ARM mejora otras arquitecturas RISC y ofrece como ventajas la reducción del área de silicio, procesador más simple, menor tamaño de código, menor consumo de potencia y, en consecuencia, incrementa el rendimiento. La Figura 4.5 muestra la estructura de una arquitectura ARM básica con memoria cache separada para instrucciones y datos.

Las principales características que aporta ARM son: (1) Reducido número de instrucciones. (2) La ejecución de las instrucciones se divide en unidades que pueden ser procesadas en paralelo. (3) Posee un amplio conjunto de registros de propósito general que pueden contener tanto datos como direcciones. (4) Es una arquitectura de carga-almacenamiento y al tener los datos en los registros es posible reducir los accesos a memoria, el número de instrucciones y que estas sean más simples.

En la Figura 4.6 se muestran los bancos de registros para un procesador de la familia ARM9 [164], donde el número de bancos está asociado a los modos de trabajo del procesador.

Otras características han permitido mejorar el rendimiento, como pasar de una ventana de registros de 32 a 16[8], reducción del retardo asociado a las instrucciones de salto que producían paradas e interrumpían el flujo de instrucciones y sobre todo, incrementar el número de instrucciones cuya ejecución se aproxime a un sólo ciclo de reloj.

Desde la primera versión, han ido apareciendo sucesivas versiones mejoradas.

[7] *Pipeline* es una técnica que segmenta la cpu en diferentes etapas, donde la salida de una es la entrada de la siguiente. El objetivo es incrementar el paralelismo y la velocidad de ejecución de las instrucciones.

[8] Una ventana de 32 registros ocupa un área mayor en el chip e incrementa el coste.

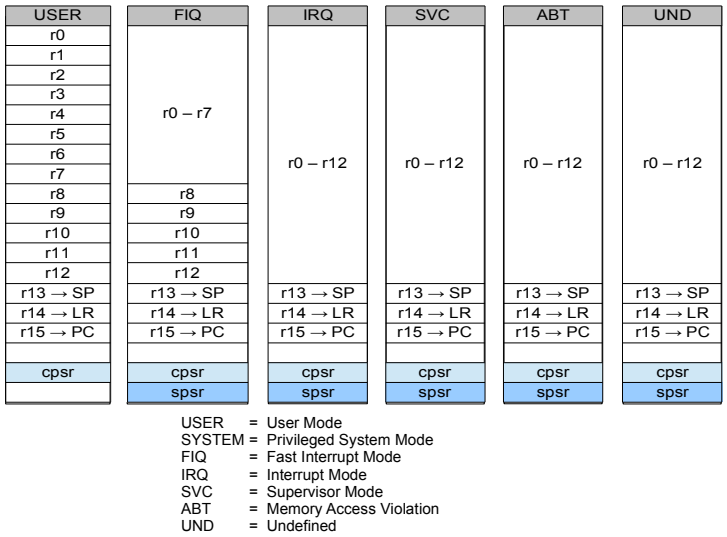


Figura 4.6: Bancos de registros de un procesador de la arquitectura ARM9, basada en RISC y según los 7 modos del procesador [19]. El sistema utiliza los mismos registros que el modo usuario, pero en modo privilegiado.

En octubre de 2015, la última familia añadida a las ya existentes es ARMv8 con dos variantes, ARMv8-A para alto rendimiento en el ámbito empresarial y móviles y ARMv8-R especialmente diseñada para aplicaciones en sistemas empotrados en el ámbito del control industrial y de la automoción. Estas arquitecturas está disponibles tanto para 32 bits, como para 64 bits. El diseño de los procesadores ARM está dirigido a los sistemas empotrados y dado que la potencia se suministra mediante baterías, su pequeño tamaño está orientado a un bajo consumo de potencia.

A continuación, en la sección 4.4 se detalla la metodología experimental utilizada en el análisis térmico del banco de registros de estas dos arquitecturas .

4.4. Metodología

En la sección 4.1 se han definido los objetivos propuestos. En primer lugar, estimar el incremento de temperatura debido al número de accesos que se producen al banco de registros durante la ejecución de las aplicaciones; en segundo lugar, analizar mediante un MOEA si es posible adaptar la distribución espacial del banco de registros, de acuerdo a las necesidades de cada tipo de aplicación y reducir el impacto térmico. La separación física de los registros con mayor número de ac-

cesos tiene como resultado una disminución en el incremento de temperatura. Por tanto, la reasignación adecuada de los registros lógicos a su registro físico permitirá optimizar su temperatura y consumo. Para ello, se ha tomado como referencia el banco de registros de las arquitecturas ARM y VLIW, presentadas en las secciones anteriores.

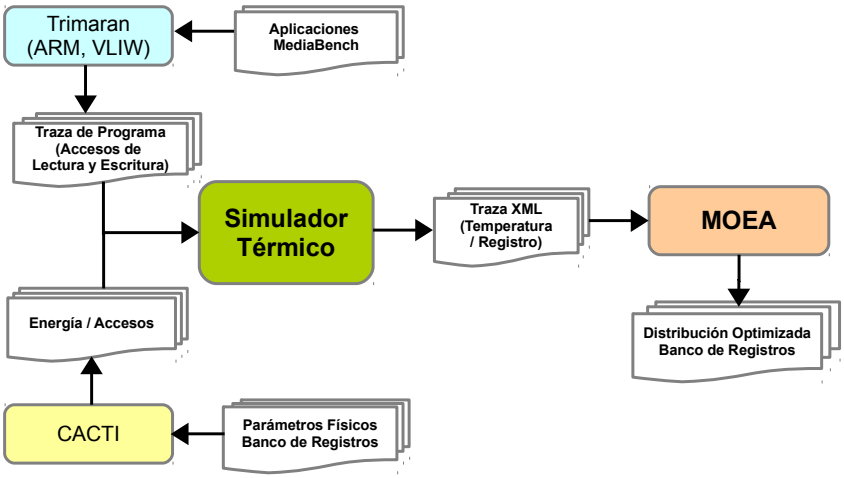


Figura 4.7: Metodología para abordar el proceso de optimización. En primer lugar, se simulan las aplicaciones mediante Trimaran para obtener una traza, que contiene los accesos a registro. Mediante CACTI, se calcula el consumo energético, dependiente de las propiedades físicas del banco de registros. Estos datos son procesados por el simulador térmico diseñado en Matlab para obtener, según el modelo térmico dado, la temperatura máxima alcanzada por cada registro y, por tanto, por el banco de registros. Tras analizar los datos de la traza y gráficos de temperatura, un MOEA permite distribuir los accesos a través del banco de registros y minimizar el impacto térmico.

Para conseguir los objetivos, es fundamental la simulación del diseño físico, en este caso del banco de registros. Así, tamaño del banco de registros, número de celdas y tamaño de celda son parámetros necesarios para adoptar un modelo multigrid que deberá ser adaptado según la arquitectura a utilizar. En esta tesis, se toma como base el diseño físico propuesto en [165], que divide el área de diseño en celdas unitarias y que debe ser modificado, de acuerdo al tamaño objetivo. El tamaño del banco de registros se ha parametrizado dependiendo de la arquitectura y se ha utilizado el mismo tamaño en cuanto al número de celdas y tamaño de celda por registro.

En el modelo propuesto, un registro está formado por 3 celdas de alto y 90 de

ancho y cada celda tiene un tamaño de $9\mu m^2$ ($3\mu m \times 3\mu m$). La Tabla 4.1 detalla para cada arquitectura las medidas físicas del banco de registros, de acuerdo a una distribución de registros en una única columna. Las medidas son expresadas en micras y en celdas.

Tabla 4.1: Especificaciones físicas del banco de registros.

Parámetros	VLIW	ARM
Nº de Registros	32	16
Anchura en celdas	90	90
Altura en celdas	96	48
Tamaño celda ($ancho \times alto$) (μm)	3×3	3×3
Anchura Banco Registros en μm	270	270
Altura Banco Registros en μm	288	144

El segundo paso consiste en obtener los datos necesarios para realizar las estimaciones en cuanto a temperatura y potencia. De este modo, hay que seleccionar un conjunto de aplicaciones a ejecutar en ambas arquitecturas, VLIW y ARM, simular dichas aplicaciones para poder obtener el número de accesos a cada registro y con ello calcular la energía consumida en cada registro.

Para hacer el seguimiento del número de accesos al banco de registros, se ha optado por la utilización del software Trimaran [166], ampliamente utilizado en el ámbito científico para la obtención de múltiples métricas sobre la ejecución de aplicaciones [44, 45, 167]. Trimaran hace uso de diferentes programas para la simulación de ARM y VLIW. En el caso de ARM se basa en el programa *sim-outorder*, perteneciente al conjunto de herramientas de SimpleScalar [168], mientras que para VLIW utiliza su propio entorno de simulación. Teniendo esto en cuenta, ambos programas se han modificado y adaptado a los requisitos para la obtención de un traza de programa que proporcione información sobre los accesos por registro y sea capaz de agruparlos según el tipo de acceso (lectura y escritura). Una traza de programa es una secuencia que guarda información sobre las diferentes operaciones que realiza una aplicación durante su ejecución, en este caso de los accesos que se realizan al banco de registros. En este sentido, cada aplicación se simula utilizando

Trimaran y durante la simulación todos los accesos a registros y su tipo se almacena en el fichero de traza. La generación del fichero de traza es un proceso que se realiza de forma *off-line* y sólo es necesario realizarlo una vez.

Para obtener la información referente a la cantidad de energía consumida en cada acceso se ha utilizado la herramienta CACTI [52], que permite estimar la energía consumida para las diferentes estructuras del procesador, mediante la especificación de sus parámetros físicos (memorias cache, memoria RAM, bancos de registros, etc). Al igual que sucede con el fichero de traza, la caracterización del banco de registros es un proceso *off-line* que se realiza una sóla vez.

```
<Floorplan Version="5.0" CellSize="3" Length="90" Width="48" NumLayers="1" NumPowerProfiles="1">
<Blocks>
<id="0" name="GPR0" type="0" xMin="0" x="0" xMax="0" yMin="0" y="0" yMax="45" zMin="0" z="0" zMax="0" l="90" w="3" h="1" p="0.67318"/>
<id="1" name="GPR1" type="0" xMin="0" x="0" xMax="0" yMin="0" y="3" yMax="45" zMin="0" z="0" zMax="0" l="90" w="3" h="1" p="0.09377"/>
<id="2" name="GPR2" type="0" xMin="0" x="0" xMax="0" yMin="0" y="6" yMax="45" zMin="0" z="0" zMax="0" l="90" w="3" h="1" p="0.02892"/>
<id="3" name="GPR3" type="0" xMin="0" x="0" xMax="0" yMin="0" y="9" yMax="45" zMin="0" z="0" zMax="0" l="90" w="3" h="1" p="0.02708"/>
<id="4" name="GPR4" type="0" xMin="0" x="0" xMax="0" yMin="0" y="12" yMax="45" zMin="0" z="0" zMax="0" l="90" w="3" h="1" p="0.14851"/>
<id="5" name="GPR5" type="0" xMin="0" x="0" xMax="0" yMin="0" y="15" yMax="45" zMin="0" z="0" zMax="0" l="90" w="3" h="1" p="0.05825"/>
<id="6" name="GPR6" type="0" xMin="0" x="0" xMax="0" yMin="0" y="18" yMax="45" zMin="0" z="0" zMax="0" l="90" w="3" h="1" p="0.05213"/>
<id="7" name="GPR7" type="0" xMin="0" x="0" xMax="0" yMin="0" y="21" yMax="45" zMin="0" z="0" zMax="0" l="90" w="3" h="1" p="0.07555"/>
<id="8" name="GPR8" type="0" xMin="0" x="0" xMax="0" yMin="0" y="24" yMax="45" zMin="0" z="0" zMax="0" l="90" w="3" h="1" p="0.01221"/>
<id="9" name="GPR9" type="0" xMin="0" x="0" xMax="0" yMin="0" y="27" yMax="45" zMin="0" z="0" zMax="0" l="90" w="3" h="1" p="0.32940"/>
<id="10" name="GPR10" type="0" xMin="0" x="0" xMax="0" yMin="0" y="30" yMax="45" zMin="0" z="0" zMax="0" l="90" w="3" h="1" p="0.13822"/>
<id="11" name="GPR11" type="0" xMin="0" x="0" xMax="0" yMin="0" y="33" yMax="45" zMin="0" z="0" zMax="0" l="90" w="3" h="1" p="0.00435"/>
<id="12" name="GPR12" type="0" xMin="0" x="0" xMax="0" yMin="0" y="36" yMax="45" zMin="0" z="0" zMax="0" l="90" w="3" h="1" p="0.00227"/>
<id="13" name="GPR13" type="0" xMin="0" x="0" xMax="0" yMin="39" y="39" yMax="39" zMin="0" z="0" zMax="0" l="90" w="3" h="1" p="0.02619"/>
<id="14" name="GPR14" type="0" xMin="0" x="0" xMax="0" yMin="42" y="42" yMax="42" zMin="0" z="0" zMax="0" l="90" w="3" h="1" p="0.00407"/>
<id="15" name="GPR15" type="0" xMin="0" x="0" xMax="0" yMin="45" y="45" yMax="45" zMin="0" z="0" zMax="0" l="90" w="3" h="1" p="0.00399"/>
</Blocks>
</Floorplan>
```

Figura 4.8: Ejemplo de fichero XML, donde cada línea se corresponde con un registro que tiene asociada una etiqueta, un nombre, su posición dada por las coordenadas X e Y respecto al punto de referencia del área de diseño, anchura, altura y densidad de potencia calculada en función de las lecturas y escrituras.

Una vez generada la traza de cada aplicación con todos los accesos realizados a cada registro durante su ejecución, se convierte a un formato XML personalizado, para facilitar el tratamiento posterior. La Figura 4.8 muestra un ejemplo de fichero XML, donde se observa que cada registro tiene asociada una etiqueta, un nombre, su posición dada por las coordenadas X e Y (en el área de diseño), anchura, altura y densidad de potencia. De esta forma, para cada arquitectura objetivo y cada aplicación se dispone de una traza que se ajusta a los requisitos del proceso de optimización. Durante este proceso, a cada registro se le asigna una posición dentro del área establecida y se calcula la potencia consumida por cada uno de ellos durante un número de ciclos establecidos para la simulación. La densidad de potencia de un

registro se define como la potencia consumida por un registro dividido por el área del registro:

$$dp = P_{\text{reg}}/A_{\text{reg}} \tag{4.7}$$

Donde dp es la densidad de potencia, P_{reg} la potencia consumida por un registro y A_{reg} es el área del registro. El cálculo de P_{reg} se rige por la Ecuación (4.8), donde R y W son el número de accesos de lectura y escritura, respectivamente; E_{r} y E_{w} la energía consumida en cada acceso de lectura y escritura; N_{c} es el número de ciclos que ha tardado la aplicación en ejecutarse y T_{c} representa el tiempo que dura un ciclo $T_{\text{c}} = 1/F$, con F como la frecuencia de trabajo.

$$P_{\text{reg}} = \frac{R \times E_{\text{r}}}{N_{\text{c}} \times T_{\text{c}}} + \frac{W \times E_{\text{w}}}{N_{\text{c}} \times T_{\text{c}}} \tag{4.8}$$

La traza XML generada sirve como entrada al simulador térmico que se ha desarrollado en Matlab para esta tesis y que se basa en el modelo térmico en estado estacionario, previamente mencionado, para obtener la temperatura. A partir de los resultados que proporciona el simulador térmico, un vector de registro/temperatura, y la representación gráfica de los valores térmicos, se pueden observar los puntos calientes dentro del banco de registros y en base a ellos, analizar la necesidad o no de optimización.

Este proceso se muestra gráficamente en la Figura 4.7, donde se puede ver la metodología utilizada en cada paso, la obtención de las trazas de las aplicaciones con el software Trimaran, la caracterización del banco de registro con CACTI, la aplicación del simulador térmico y, por último, la ejecución del MOEA.

El proceso de optimización mencionado se lleva a cabo con el Algoritmo Evolutivo Multiobjetivo NSGA-II, concretamente bi-objetivo, con el que se simula la modificación de la estructura física del banco de registros según su posición relativa en el área de diseño, y según las características de la arquitectura tratada. Se ha utilizado una codificación donde el cromosoma se corresponde con una distribución geométrica determinada del banco de registros y cada registro es un alelo.

El objetivo tiene dos fases: (1) determinar la configuración geométrica de los registros y (2) determinar la posición de cada registro en esta configuración. La fase (1) está dirigida a la etapa de fabricación; la fase (2) se encarga de decirle al compilador qué registros debe usar en lugar de los que él había determinado originalmente.

La distribución espacial, permutando la asignación de los accesos a registro, se realiza con el fin de separar físicamente aquellos registros con una densidad de potencia mayor y reducir el impacto térmico. Los objetivos a optimizar se definen como la minimización del impacto térmico y la viabilidad física del área diseñada, es decir que todos los registros se encuentran dentro del área de diseño y cada uno en una posición independiente. Hay que tener en cuenta que dentro de cada arquitectura, algunos registros son utilizados para propósitos específicos (puntero de pila, contador de programa, etc.), generalmente la posición relativa de estos registros no puede ser alterada. De esta forma, las restricciones que impone cada arquitectura deben ser proporcionadas al MOEA, a través de la traza generada. Estas restricciones se marcan en el fichero de traza asignando una posición fija al registro dentro de la configuración geométrica determinada.

El esquema general del algoritmo NSGA-II se muestra en la Figura 3.9 en el Capítulo 2. Para el problema de optimización que se presenta en este capítulo de la tesis, un individuo de la población representa una configuración del banco de registros, es decir una distribución espacial de los registros. Esta configuración debe contener todos los registros de acuerdo a la arquitectura objetivo y las restricciones ya mencionadas según la propia arquitectura.

La aplicación de los operadores genéticos de cruce y mutación debe dar como resultado individuos que cumplan con los requisitos mencionados. La configuración del MOEA se muestra en la Tabla 4.2. Las probabilidades de cruce y mutación son las recomendadas en [150].

Un cromosoma representa una distribución espacial de los registros dentro de un área de diseño dada. El cromosoma se forma con una secuencia de valores, donde cada valor identifica una posición dentro de la distribución espacial que representa.

Tabla 4.2: Parámetros del algoritmo MOEA.

Número de generaciones	250
Tamaño de la población	100
Longitud del cromosoma	N_{reg}
Probabilidad de cruce	0,9
Probabilidad de mutación	$1/N_{reg}$

La Figura 4.9 muestra un ejemplo de cromosoma cuya longitud, por simplicidad, se ha reducido a 8, distribuidos en una columna. $R5$ corresponde a los accesos del registro 5 que se han trasladado a la primera posición, $R2$ los accesos al registro 2 que ahora ocupan la segunda posición, y así sucesivamente.

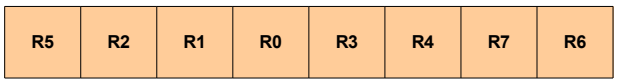


Figura 4.9: Ejemplo de cromosoma para un banco de registros con 8 registros distribuidos en una columna. Cada gen del cromosoma representa la posición dentro de la distribución espacial.

La Figura 4.10 representa gráficamente la aplicación de los operadores genéticos (selección, cruce y mutación). En cada generación se seleccionan por torneo dos cromosomas de la población. En cada ciclo se generan dos hijos utilizando cruce cíclico (*Crossover Cycle*). Cada ciclo comienza con el primer gen del cromosoma 1, el cual se busca en el cromosoma 2 (proceso representado por las flechas de color negro), una vez encontrado se mira la misma posición en el cromosoma 1 (operación representada por las flechas de color rojo), y se busca dicho gen en el cromosoma 2, el proceso se repite hasta llegar a una posición que ya se ha recorrido. Las posiciones no accedidas son las que se intercambian entre ambos cromosomas (proceso representado por las flechas de color verde), obteniendo dos descendientes nuevos. Para aplicar mutación, se ha seleccionado el operador de mutación de enteros clásico (*integer-flip*), con una probabilidad que depende del número de registros de la arquitectura N_{reg}, es decir, del tamaño del cromosoma (1/32 para VLIW y 1/16 para ARM).

La evaluación de los individuos se lleva a cabo mediante una función de coste

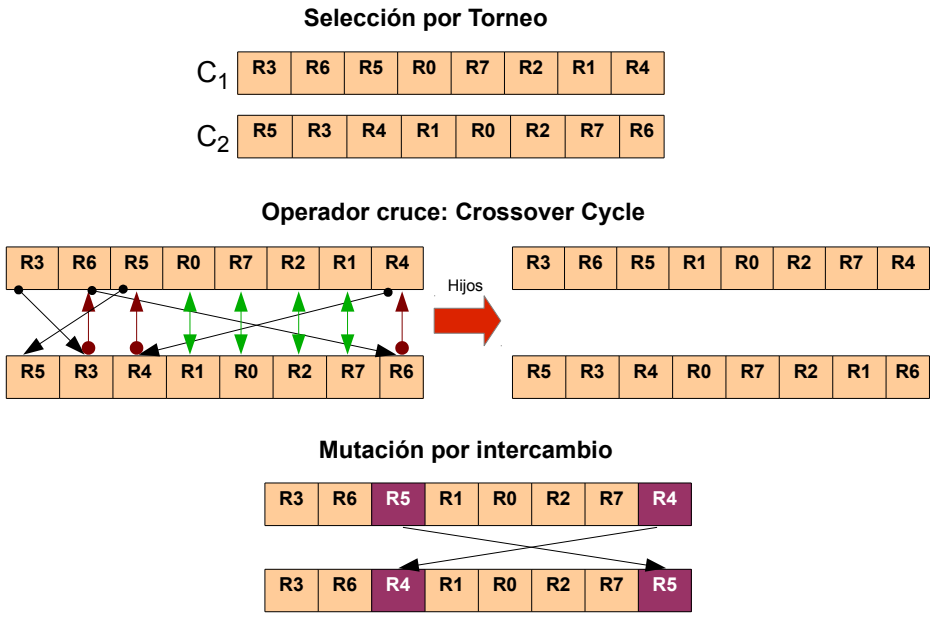


Figura 4.10: Ejemplo de aplicación de los operadores genéticos. Selección por torneo: se seleccionan dos cromosomas de la población que compiten entre sí, el mejor será seleccionado. El proceso se repite para conseguir dos cromosomas para aplicar la operación de cruce. Se aplica cruce cíclico como operador de cruce entre los dos cromosomas. El proceso se inicia en el gen 1 del cromosoma 1, el cual se busca en el cromosoma 2 (proceso representado por las flechas de color negro), una vez encontrado se mira la misma posición en el cromosoma 1 (operación representada por las flechas de color rojo), y se busca dicho gen en el cromosoma 2, el proceso se repite hasta llegar a una posición que ya se ha recorrido. Las posiciones no accedidas serán las que se intercambiarán entre ambos cromosomas (proceso representado por las flechas de color verde), obteniendo dos descendientes nuevos. Para mutación se aplica *integer-flip* con una probabilidad de $1/N_{reg}$

.

que asignará un valor a cada individuo y que debe evaluar dos objetivos: (1) La viabilidad física de la distribución de los registros, es decir, que todos se encuentran dentro del área de diseño y ocupan un espacio independiente, los valores X e Y de cada registro permiten controlar su posición. (2) El impacto térmico de la configuración propuesta que debe de analizar la densidad de potencia de cada registro y la influencia que ejercen sobre él los registros que le rodean. La función objetivo para el impacto térmico queda definida en el Ecuación (4.9).

$$f = \sum_{i=1}^{N_{reg}} \frac{(dp_i \times dp_j)}{d_{ij}} \tag{4.9}$$

donde dp_i y dp_j son la densidad de potencia de los registros i, j y d_{ij} es la

distancia euclídea[9] entre ellos.

La optimización mediante un MOEA proporciona una o varias soluciones, denominadas aproximaciones al Frente Óptimo de Pareto y para cada una de ellas se ejecuta el simulador térmico para medir el impacto térmico de dichas soluciones sobre la distribución original. Las soluciones se comparan con el fin de concluir si la nueva disposición geométrica reduce el impacto térmico.

4.5. Pruebas experimentales

En la sección 4.4 se ha explicado el proceso seguido y las herramientas utilizadas para el análisis térmico, así como el proceso de optimización. Esta sección muestra un resumen de las pruebas realizadas y analiza los resultados obtenidos. Para las pruebas experimentales se han utilizado las aplicaciones especificadas en la Tabla 4.3 que es un subconjunto perteneciente al conjunto de programas Mediabench [169], principalmente programas multimedia.

Tabla 4.3: Conjunto de programas Mediabench utilizados.

Nombre	**Descripción**
epic-unepic	Programas experimentales de compresión-descompresión de imágenes.
cjpeg-djpeg	Programas de compresión-descompresión estándar de imágenes y pensado, entre otros, para imágenes del mundo real.
gsmdecode-gsmencode	Programas de compresión-descompresión de audio.
rawcaudio-rawdaudio	Programas de compresión-descompresión de voz.
mpegdecode-mpegencode	Programas de codificación-decodificación de vídeo que enfatiza en la correcta aplicación del estándar MPEG.

Para realizar los experimentos se ha propuesto, en cada una de las arquitecturas previamente mencionadas, el diseño de tres posibles configuraciones definidas por la distribución de los registros sobre el área de diseño. Cada configuración viene definida por el número de filas y columnas, por tanto por la topología de las celdas que conforman cada registro. Por ejemplo, en una configuración de 32 x 1 para la arquitectura VLIW los registros se distribuyen en 32 filas y 1 columna, mientras

[9]Distancia euclídea entre dos puntos a y b, donde $a = (x_a, y_a)$ $b = (x_b, y_b)$, se define como $d(a, b) = \sqrt{(x_b - x_a)^2 + (y_b - y_a)^2}$

que para una configuración con 2 columnas los registros se distribuyen en 16 filas y 2 columnas. De acuerdo a las especificaciones indicadas en la Tabla 4.1, el tamaño de una fila es de 3 celdas de alto por 90 de ancho. De esta forma, las celdas de cada registro deben ubicarse correctamente en el área establecida siguiendo una distribución de las celdas por columnas, para la configuración específica. La Tabla 4.4 muestra la parametrización, en cuanto al número de filas y columnas en el que se distribuyen las celdas que forman el banco de registros, de acuerdo a cada configuración abordada.

Tabla 4.4: Especificaciones físicas banco de registros.

Arquitectura VLIW					**Arquitectura ARM**				
Distribución			**Celdas**		**Distribución**			**Celdas**	
Conf.	Filas	Columnas	Ancho	Alto	Conf.	Filas	Columnas	Ancho	Alto
C1	32	1	90	96	C1	16	1	90	48
C2	16	2	180	48	C2	8	2	180	24
C3	4	8	720	12	C3	2	8	360	6

A continuación se detalla la distribución original para cada arquitectura y configuración. Para VLIW $C1$ (32 filas $\times$1 columna), los registros se encuentran en orden natural de $R0$ a $R31$; $C2$ (16 filas $\times$ 2 columnas), los registros se ubican por columnas, de $R0$ a $R15$ primera columna y de $R16$ a $R31$ en la segunda; $C3$ (4 filas $\times$ 8 columnas), los registros se ubican de $R0$ a $R3$ en la primera columna hasta $R28$ a $R31$ en la octava columna. En el caso de ARM, $C1$ (16 filas $\times$ 1 columna), los registros aparecen en orden natural de $R0$ a $R15$; $C2$ (8 filas $\times$ 2 columnas), los registros se ubican por columnas, de $R0$ a $R7$ primera columna y de $R8$ a $R15$ en la segunda; $C3$ (2 filas $\times$ 8 columnas), los registros se sitúan de $R0$ a $R1$ en la primera columna hasta $R14$ a $R15$ en la octava columna.

4.5.1. Simulación de aplicaciones

El conjunto de pruebas experimentales se ha llevado a cabo para las tres configuraciones propuestas en cada arquitectura y las aplicaciones descritas en la Tabla 4.3. Cada aplicación se ha simulado de manera independiente y se ha estudiado la in-

fluencia de cada una sobre el banco de registros. El análisis del impacto térmico se ha realizado calculando para cada registro la temperatura media, a través de la temperatura de las celdas que componen cada registro; la temperatura máxima, como la máxima temperatura alcanzada por las celdas que lo componen y la potencia consumida por cada registro. Todas las gráficas, que se muestran a continuación, trabajan con una escala de incremento sobre la temperatura ambiente de entre 0 y $5,3044\ ^{o}C$, la cual corresponde al incremento de temperatura máximo alcanzado durante los experimentos.

Tomando como base una arquitectura VLIW básica, las figuras 4.11, 4.12, 4.13 muestran la representación térmica de las celdas que componen cada registro (3 celdas de alto x 90 de ancho), para las configuraciones $C1$, $C2$ y $C3$, respectivamente. Cada celda se representa con la temperatura calculada, según el modelo térmico previamente descrito, tras la simulación de las aplicaciones y la obtención del número de accesos. En la Figura 4.11 los 32 registros se distribuyen dentro del banco de registros en 32 filas x 1 columna, en la Figura 4.12, la disposición de los registros es en 16 filas y 2 columnas, finalmente la Figura 4.13 presenta una distribución de 4 filas y 8 columnas. El incremento máximo de temperatura se sitúa en $0,4319^{o}C$ para las configuraciones estructurales $C1$ y $C2$, y $0,4318^{o}C$ para la configuración $C3$, todos ellos alcanzados en la aplicación *gsmencode*. El incremento de temperatura medio se corresponde con los valores $0,3966^{o}C$ para $C1$ y $C2$, y $0,3974^{o}C$ para $C3$.

Teniendo en cuenta que dos de los factores que definen el comportamiento térmico del banco de registros son el número de accesos soportados por los registros que lo componen y la temperatura de los registros vecinos (el resto de componentes que rodean el banco de registros y su influencia sobre éste, no se estudian en esta tesis), en las tres gráficas se puede observar cómo los registros exteriores tienen una temperatura menor que los registros interiores. Debido a la interacción existente entre registros vecinos que presentan diferentes temperaturas, se produce una transferencia de calor por conducción y siempre en sentido negativo de los registros con mayor temperatura a los que poseen menor temperatura. En un estudio térmico en

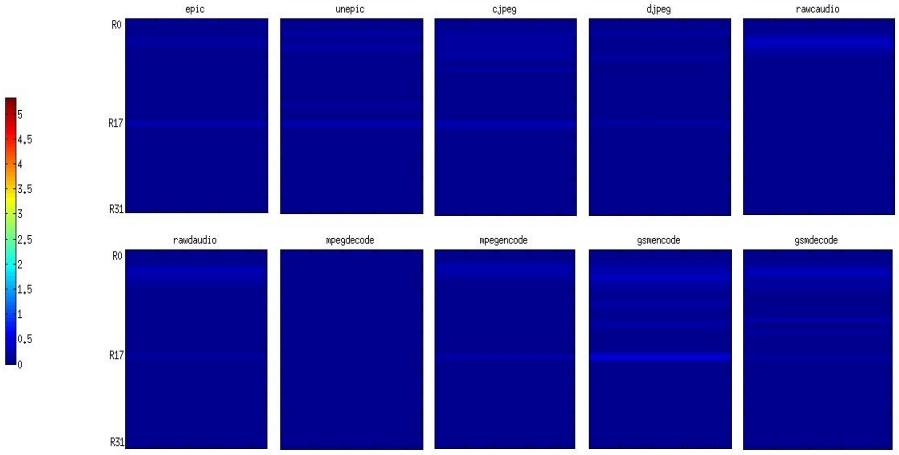


Figura 4.11: Incremento de temperatura debido a los accesos al banco de registros,correspondientes a los registros de propósito general de una arquitectura VLIW básica. Se muestra la configuración $C1$, con 32 registros dispuestos en una única columna, donde cada registro tiene 3 celdas de alto y 90 de ancho. Cada celda del banco de registros es representada con el incremento de temperatura alcanzado. En esta configuración, los incrementos de temperatura máximo y medio se sitúan en $0,4319^{o}C$ y $0,3966^{o}C$, respectivamente.

modo estacionario, como es el caso, esta distribución es lineal y puede ser expresada mediante la Ley de Fourier, tal y como ha quedado expuesto en la sección 4.2.

El incremento de la densidad de potencia lleva a un incremento de la temperatura pico y, por otra parte, en VLIW la frecuencia de accesos al banco de registros es elevada debido principalmente al empaquetamiento de varias instrucciones en una de mayor longitud. Además, las aplicaciones presentan diferente comportamiento dependiendo de la fase de ejecución, por tanto, el acceso al banco de registros tampoco es uniforme y aparecen puntos calientes debido a la ejecución de bucles o llamadas sucesivas a funciones. Sin embargo, las técnicas de compilación desarrolladas en los últimos años como la reasignación de nombres de registros o reasignación de registros según el rango de vida de las variables permiten reducir la densidad de potencia y distribuir los accesos a lo largo del banco de registros y, en consecuencia, evitar que la temperatura alcance valores demasiado altos en las fase de las aplicaciones donde la actividad es más intensa. En las pruebas experimentales para VLIW, los accesos se distribuyen a lo largo del banco de registros y aquellos con un incremento de temperatura mayor en todos las aplicaciones, se corresponden con

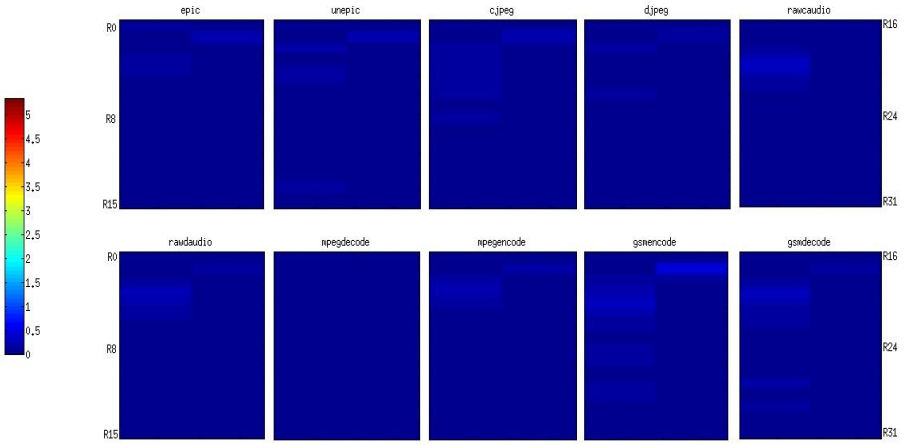


Figura 4.12: Incremento de temperatura debido a los accesos al banco de registros, correspondientes a los registros de propósito general de una arquitectura VLIW básica. Se representa la configuración $C2$, con 32 registros dispuestos en dos columnas ($R0...R15$ y $R16....R31$ para la primera y segunda columna respectivamente) y, por tanto, de 48 x 180 celdas. En esta configuración, los incrementos de temperatura máximo y medio se sitúan en $0,4318^oC$ y $0,3966^oC$, respectivamente.

los registros globales ($R0 - R7$), de salida ($R8 - R15$) y el puntero de pila ($R17$), utilizado en la llamada a procedimientos/funciones y almacenamiento/recuperación de datos.

A continuación se pone el foco de atención en la arquitectura ARM. Las figuras 4.14, 4.15, 4.16 representan el impacto térmico sobre el banco de registros de la arquitectura ARM objetivo con el tamaño de celda y de registro anteriormente mencionados y según las configuraciones estructurales propuestas $C1$, $C2$ y $C3$, respectivamente. La Figura 4.14 representa el impacto térmico para la configuración $C1$ con una distribución de registros de 16 x 1, La Figura 4.15, presenta la disposición de 8 x 2, correspondiente a la configuración $C2$, mientras que en la Figura 4.16 se muestra la distribución de 2 x 8 (la figura aparece en un formato 8 x 2, con el fin de mantener el formato de representación utilizado en el resto de gráficas), haciendo referencia a una disposición donde los registros se distribuyen por filas y columnas. El incremento máximo de temperatura es de $5,3044^oC$ para $C1$ y $C2$, y $5,3036^oC$ para $C3$, alcanzados por las aplicaciones $cjpeg$ y $djpeg$, respectivamente. El incremento medio de temperatura se sitúa en $4,7932^oC$, $4,79969^oC$ y $4,7963^oC$ para

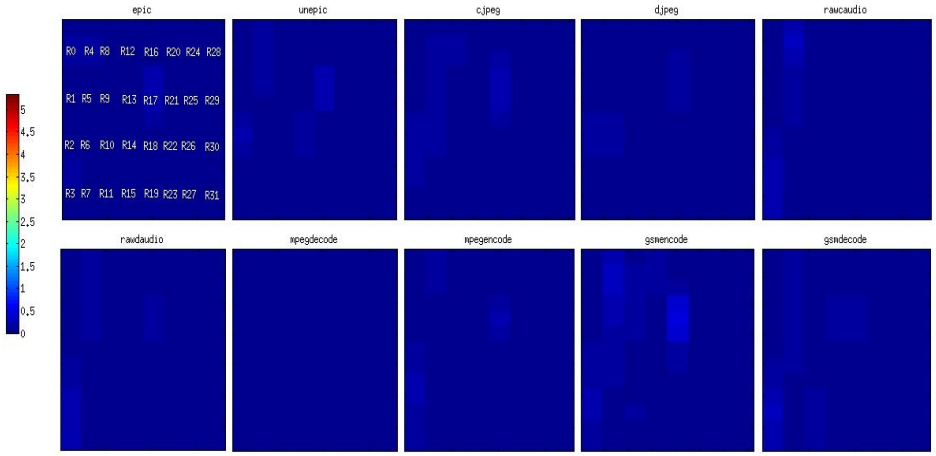


Figura 4.13: Incremento de temperatura debido a los accesos al banco de registros, correspondientes a los registros de propósito general de una arquitectura VLIW básica. La figura muestra el estudio térmico correspondiente a la configuración $C3$, con 32 registros dispuestos en 4 filas y 8 columnas (columna 1: $R0..R3$, columna 2: $R4..R7$, columna 3: $R8..R12$ y así, sucesivamente), con una distribución de celdas por registro de 3 celdas de alto y 90 de ancho (12 x 720 celdas). En esta configuración, los incrementos de temperatura máximo y medio se sitúan en $0,4318^oC$ y $0,3974^oC$, respectivamente.

$C1$, $C2$ y $C3$, respectivamente.

Es en la configuración $C3$ para cada modelo de arquitectura donde se observa de forma clara como las celdas interiores presentan un impacto térmico mayor debido a la influencia de las celdas que las rodean. A pesar de la baja influencia detectada tanto sobre VLIW como ARM, se considera la realización del proceso de optimización. Esta optimización consiste en obtener una distribución espacial de los registros, adaptada al tipo de aplicación a ejecutar, mediante un MOEA y que permita reducir el incremento de temperatura que los continuos accesos al banco de registros suponen.

La baja influencia detectada indica que las técnicas aplicadas por el compilador permiten mantener la temperatura del banco de registros dentro de un rango aceptable. En el caso del compilador Trimaran, realiza tanto optimizaciones clásicas (propagación de constantes, eliminación de código muerto y subexpresiones comunes), como rotación de registros y desenrollado de bucles, entre otros. Respecto a ARM, en todo momento hay visibles 15 registros, los registros $R0 - R7$, también

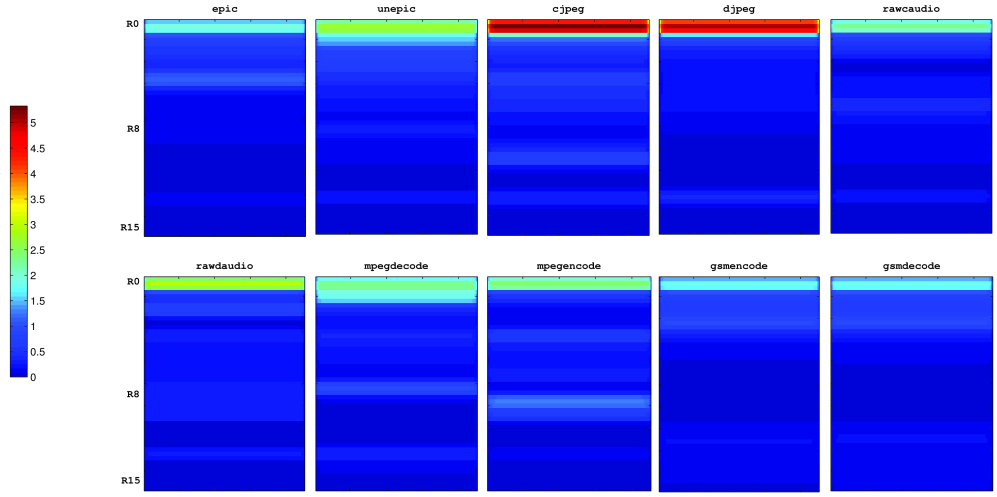


Figura 4.14: Impacto térmico de los accesos al banco de registros, correspondientes a los registros de propósito general de una arquitectura ARM básica. La configuración $C1$, con una distribución de 16 registros dispuestos en una única columna y un tamaño de celda de 3 celdas de alto y 90 de ancho. La gráfica térmica representa el incremento de temperatura del banco de registros, a partir del incremento de temperatura de los registros que lo forman y basado en el incremento de temperatura de cada una de sus celdas. El incremento máximo y medio de temperatura se sitúa en $5,3044^oC$ y $4,7932^oC$, respectivamente.

conocidos como $Lo-Registers$ son utilizados por cualquier instrucción, es por ello por lo que son los que mayor número de accesos presentan y, por consiguiente, mayor incremento de temperatura. $R8-R12$ o $Hi-Registers$ sólo son usados por un conjunto reducido de instrucciones, mientras que $R13$ es el puntero de pila SP, $R14$ se corresponde con $Link-Register(LR)$ y $R15$ es el contador de programa o CP.

Los incrementos de temperatura alcanzados tanto para VLIW, como ARM, no son altos, lo cual nos lleva a concluir que el proceso de optimización no aporta un beneficio significativo. No obstante, se realiza un análisis pormenorizado de los datos debido a que en porcentajes, se obtienen algunos resultados que consideramos dignos de estudio. La siguiente sección presenta los resultados obtenidos tras este proceso de optimización.

4.5.2. Optimización del banco de registros

El proceso de optimización, explicado en la sección 4.4, se lleva a cabo de forma individual para cada aplicación. El proceso es conducido por el algoritmo NSGA-II,

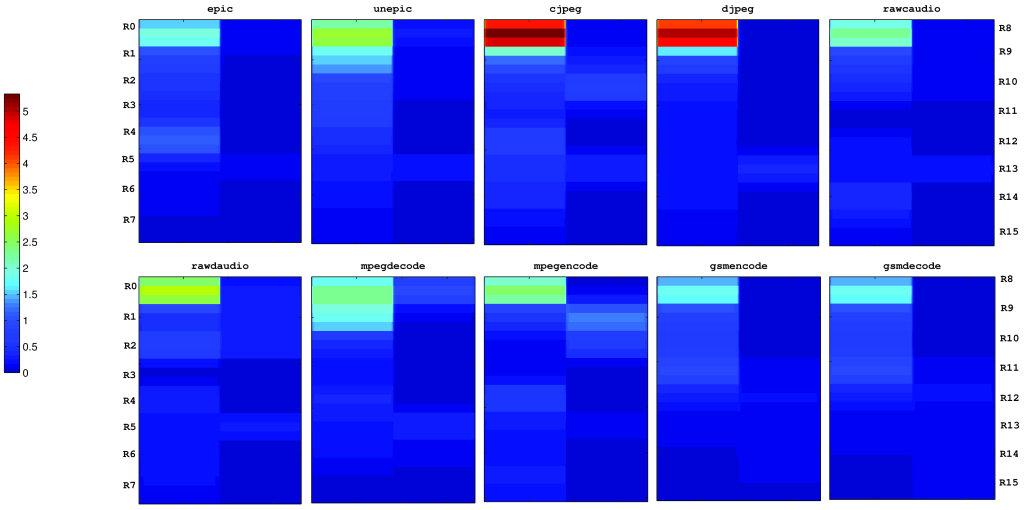


Figura 4.15: Impacto térmico debido a los accesos al banco de registros, correspondientes a los registros de propósito general de una arquitectura ARM básica. La configuración $C2$ tiene una distribución de registros de 8 x 2 y un tamaño de celda de 3 celdas de alto y 90 de ancho. El impacto térmico está determinado por el incremento de temperatura del banco de registros, a partir del incremento de temperatura de los registros que lo forman y basado en el incremento de temperatura de cada una de sus celdas. El incremento máximo y medio de temperatura se sitúa en $5,3044^{o}C$ y $4,7969^{o}C$, respectivamente.

al cual se le proporciona una traza con la disposición física de los registros dentro del área que representa el banco de registros y la densidad de potencia de cada registro debido a los accesos soportados para la aplicación a optimizar. Conforme a las restricciones impuestas por la arquitectura y la viabilidad física, se ejecuta el algoritmo que devuelve una o varias soluciones (aproximaciones al POF). Cada solución es una configuración estructural que realiza una redistribución de los accesos a los registros según su componente térmica, de tal forma que aquellos registros con una densidad de potencia elevada se encuentren alejados entre sí y, por tanto, el incremento de temperatura sea menor. Cada solución aportada por NSGA-II se ha analizado con el fin de seleccionar aquella configuración física o distribución espacial, que proporciona una mejora más significativa. Se ha detectado que el comportamiento térmico de las soluciones pertenecientes a cada frente de Pareto es similar y, por tanto, entre las soluciones obtenidas se ha seleccionado una para cada aplicación y se ha realizado la simulación térmica que se analiza a continuación.

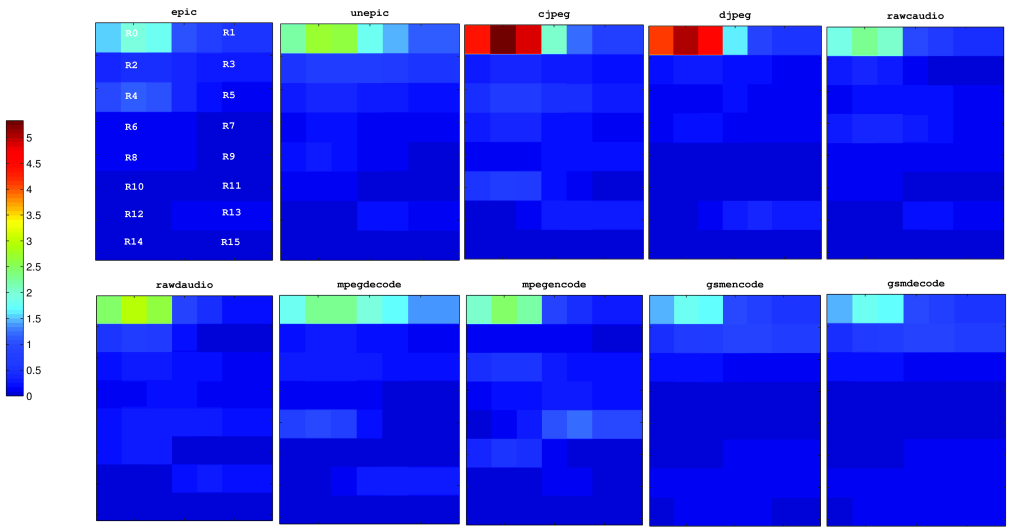


Figura 4.16: Impacto térmico debido a los accesos al banco de registros, correspondientes a los registros de propósito general de una arquitectura ARM básica. La configuración $C3$ tiene una distribución de registros de 2 x 8 y un tamaño de celda de 3 celdas de alto y 90 de ancho, aunque sólo a efectos de visualización, la figura aparece en un formato 8 x 2, con el fin de mantener el formato de representación mayoritariamente utilizado. El impacto térmico está determinado por el incremento de temperatura del banco de registros, a partir del incremento de temperatura de los registros que lo forman y basado en el incremento de temperatura de cada una de sus celdas. El incremento máximo y medio de temperatura se sitúa en $5,3036^oC$ y $4,7963^oC$, respectivamente.

4.5.2.1. Análisis global por configuración

El objetivo buscado en esta propuesta es la obtención de configuraciones que minimicen la temperatura máxima del banco de registro. El primer análisis se centra en la identificación de aquella configuración estructural, de entre las tres propuestas, que reduce de forma global la temperatura máxima. La Tabla 4.5 muestra los incrementos de temperatura medios y máximos en ^oC sobre la temperatura ambiente para cada configuración y modelo de arquitectura evaluada.

Tabla 4.5: Incrementos de temperaturas medios y máximos por configuración

Conf.	**VLIW**				**ARM**			
	No optimizado		**Optimizado**		**No optimizado**		**Optimizado**	
	Medio	Máximo	Medio	Máximo	Medio	Máximo	Medio	Máximo
C1	0,3966	0,4319	0,3878	0,4266	4,7932	5,3044	4,7049	5,2507
C2	0,3966	0,4318	0,3883	0,4266	4,7969	5,3044	4,7099	5,2507
C3	0,3974	0,4318	0,3760	0,4191	4,7963	5,3036	4,7085	5,2506

Se constata la baja influencia detectada sobre el banco de registros. A pesar de ello, también se observa una pequeña reducción de la temperatura máxima que se debe a la separación física (redistribución de accesos), de aquellos registros con mayor número de accesos y cuyos registros vecinos tienen tras la optimización un incremento menor de temperatura. Esto permite una mejor distribución de la temperatura a lo largo de la estructura física. Por tanto, el comportamiento del algoritmo de optimización es correcto.

A lo largo de la sección se habla de mejora, esta se refiere al mejor comportamiento térmico que representan las soluciones aportadas por el MOEA frente a la distribución original, de acuerdo a cada configuración abordada en este estudio. De esta forma, estas distribuciones originales son las soluciones no optimizadas mientras que las soluciones aportadas por el MOEA son las soluciones optimizadas.

El análisis pormenorizado de los datos presentados en la Tabla 4.5 permite medir el ratio de mejora conseguido como porcentaje de reducción de la temperatura máxima y de la máxima temperatura media, de forma global. Estos porcentajes aparecen reflejados en la Tabla 4.6, donde se observa que la configuración $C3$ para VLIW obtiene la reducción de la máxima temperatura media y máxima más significativa con un $5,39\,\%$ y $2,94\,\%$, respectivamente. Para ARM aunque los porcentajes de mejora para las tres configuraciones son muy similares, la configuración $C1$ se presenta como la mejor configuración global con un $1,84\,\%$ y $1,00\,\%$ para la temperatura media y máxima, respectivamente.

Tabla 4.6: Porcentajes de mejora temperaturas medias y máximas por configuración

Conf.	**VLIW**		**ARM**	
	Media %	Máxima %	Media %	Máxima %
C1	2,22	1,23	1,84	1,01
C2	2,09	1,20	1,81	1,01
C3	5,39	2,94	1,83	1,00

4.5.2.2. Distribución de temperatura por registros

Analizados los datos de forma global, se profundiza en la evolución de los valores máximos de las temperaturas medias y máximas a nivel de registro para cada configuración y teniendo en cuenta todas las aplicaciones. La Figura 4.17 muestra información relativa a la evolución de estos valores en las pruebas realizadas con las tres configuraciones posibles para VLIW. Para permitir comparar los resultados se muestra la información antes y después de la optimización mediante el MOEA.

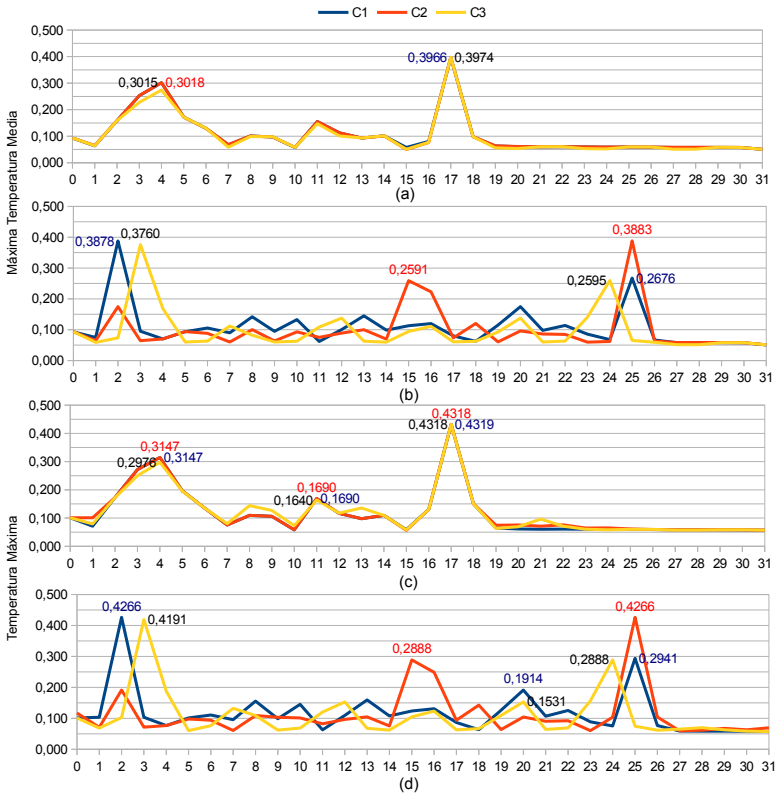


Figura 4.17: Distribución de la máxima temperatura media y máxima alcanzada, teniendo en cuenta todas las aplicaciones, debido a los accesos al banco de registros para las configuraciones estructurales estudiadas, correspondientes a los registros de propósito general de una arquitectura VLIW básica. Los valores del eje de abscisas se corresponden con la posición dentro del banco de registros, no con el número de registro. Ello es debido a que en la optimización de cada aplicación el registro que ocupa esa posición soporta el número de accesos que la solución aportada por el MOEA le asigna. La figura (*a*) representa el máximo incremento medio para cada estructura del banco de registros, teniendo en cuenta todas las aplicaciones, antes de la optimización, la figura (*b*) después de la optimización. Las figuras (*c*) y (*d*) muestran el incremento máximo por registro antes y después de la optimización mediante el MOEA. En todas ellas, se observa un comportamiento similar en cuanto al incremento de temperatura.

La figura se divide en cuatro partes, se representa el máximo incremento medio y máximo de temperatura para cada registro durante las tres configuraciones antes y después del proceso de optimización, expresado en ^{o}C. La Figura 4.17.*a* representa el máximo incremento medio de temperatura para la distribución inicial alcanzando el incremento máximo de $0,3974^{o}C$ con la configuración $C3$ y durante la aplicación *gsmencode*. Para las configuraciones $C1$ y $C2$ se observa una distribución similar de la máxima temperatura media, y no mejoran los resultados respecto a $C3$. La Figura 4.17.*b* representa el máximo incremento medio de temperatura después de la optimización mediante MOEA. La redistribución de los accesos realizada por el algoritmo de optimización permite una mejora que reduce el incremento de la temperatura media del banco de registros respecto a la distribución original. Esta mejora es más significativa en la configuración $C3$.

La Figura 4.17.*c* representa el máximo incremento de temperatura para la distribución de registros inicial, donde el máximo alcanzado es de $0,4319^{o}C$ con la configuración $C1$ y durante la aplicación *gsmencode*. $C2$ y $C3$ obtienen una distribución similar de la temperatura máxima y tampoco mejoran el resultado térmico de $C1$. La Figura 4.17.*d* presenta el máximo incremento de temperatura después de la optimización mediante MOEA. La redistribución de los accesos a través del banco de registros permite mejorar el resultado térmico de la distribución no optimizada y, por tanto, reducir el incremento de la temperatura máxima.

Mirando el resultado del algoritmo de optimización sobre las tres configuraciones abordadas, se observa que la configuración $C3$ es la que permite reducir en mayor medida el impacto térmico sobre el banco de registros. La redistribución de los accesos aportada por el MOEA alcanza una reducción del incremento de temperatura media y máxima mayor que sobre las configuraciones $C1$ y $C2$. La distribución estructural de $C3$ en 4 filas x 8 columnas, facilita la disipación de calor gracias a que un mayor número de celdas está en contacto con el exterior. Así, $C3$ mejora en un $3,04\,\%$ y un $3,18\,\%$ el máximo incremento de temperatura media, respecto a la optimización de $C1$ y $C2$, respectivamente y un $1,76\,\%$ la máxima temperatura

respecto a $C1$ y $C2$. Estos porcentajes son superiores si se compara la configuración $C3$ tras el proceso de optimización con las configuraciones $C1$ y $C2$ sin optimizar y que supone una mejora de $5,19\,\%$, en cuanto a la máxima temperatura media. Respecto a la temperatura máxima, la mejora alcanza el $2,96\,\%$ y $2,94\,\%$ en relación a $C1$ y $C2$ sin optimizar, respectivamente.

En cuanto a ARM, los porcentajes de mejora presentados en la Tabla 4.6 muestran que la optimización alcanza una mejora de $1,84\,\%$, $1,81\,\%$ y $1,83\,\%$, en relación a la reducción del máximo incremento medio de temperatura. La temperatura máxima se reduce el $1,01\,\%$ para $C1$ y $C2$, y el $1\,\%$ para $C3$. Tal y como se observa los porcentajes son reducidos, sin embargo también en este caso, se consigue optimizar respecto de la configuración original.

La distribución de las temperaturas media y máxima para los registros del banco de registros de la arquitectura ARM, se representa en la Figura 4.18, donde se observa su evolución en la distribución espacial correspondiente a las configuraciones antes y después de la optimización, para todas las aplicaciones. En el caso de ARM, a pesar de que la configuración $C3$ presenta el mayor porcentaje de mejora, la diferencia con $C1$ y $C2$ es demasiado pequeña como para situarla en una posición predominante. A diferencia de VLIW, donde realmente se postula como la mejor estructura para el banco de registros al disponer un mayor número de celdas en contacto con el exterior de la estructura, facilitando que la redistribución espacial de los accesos a registro permita una mejor disipación de calor.

Mirando los resultados desde el punto de vista de las aplicaciones, a continuación se analiza la evolución de la máxima temperatura media y la temperatura máxima. La Tabla 4.7 muestra los porcentajes obtenidos por aplicación para las configuraciones estructurales propuestas para el banco de registros de una arquitectura VLIW. En ella se observa que la redistribución de los accesos realizada por el MOEA, basada en el impacto térmico de cada registro, mejora en todos las aplicaciones en cuanto a los máximos de temperatura media y máxima alcanzados en las soluciones sin optimizar. Además, se aprecia como las configuraciones $C2$ y $C3$

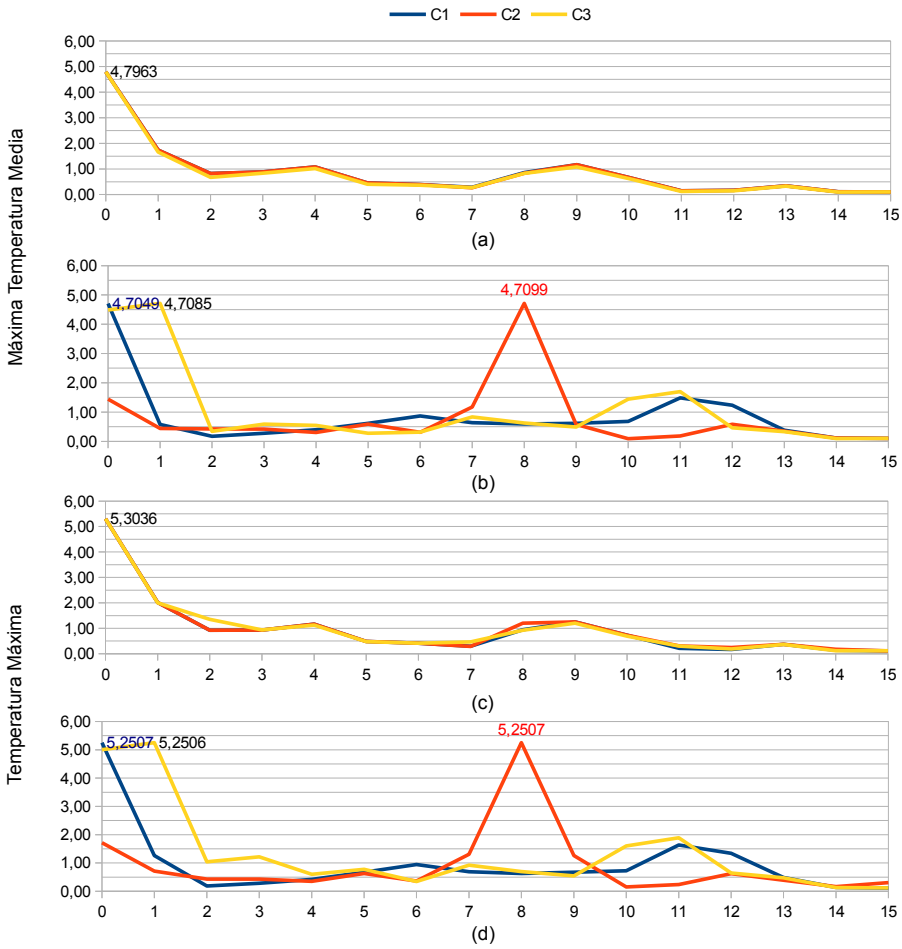


Figura 4.18: Distribución de la máxima temperatura media y máxima alcanzada, para todas las aplicaciones, a partir de los accesos al banco de registros para las configuraciones estructurales estudiadas, correspondientes a los registros de propósito general de una arquitectura ARM básica. Los valores del eje de abscisas se corresponden con la posición dentro del banco de registros, no con el número de registro. La solución optimizada para cada aplicación mediante MOEA asigna a cada posición los accesos de un registro de tal forma que se reduzca el impacto térmico. La figura (*a*) representa el máximo incremento medio para cada estructura del banco de registros, teniendo en cuenta todas las aplicaciones, antes de la optimización, la figura (*b*) después de la optimización. Las figuras (*c*) y (*d*) muestran el incremento máximo por registro antes y después de la optimización mediantes el MOEA. En todas ellas, se observa un comportamiento similar en cuanto al incremento de temperatura.

son las que presentan el mayor porcentaje de reducción térmica para la mayoría de aplicaciones.

La Tabla 4.8 presenta un resumen con los porcentajes de mejora respecto a los máximos valores alcanzados de temperatura media y máxima por aplicación y configuración estudiada para ARM. Al igual que sucede con VLIW, se puede observar

Tabla 4.7: Porcentaje de mejora respecto a máxima temperatura media y máxima por aplicación y configuración estructural para VLIW.

	epic		**unepic**		**cjpeg**		**djpeg**		**gsmdec**	
	Medio	Máximo	Medio	Máximo	Medio	Máximo	Medio	Máximo	Medio	Máximo
C1	1,97	1,11	2,0	1,14	0,54	0,30	2,04	1,15	9,49	5,47
C2	6,00	3,39	5,29	2,93	6,08	3,44	6,30	3,50	12,38	7,13
C3	6,05	3,39	5,20	2,93	6,17	3,44	6,33	3,49	3,25	1,84
	gsmenc		**rawcaudio**		**rawdaudio**		**mpegdec**		**mpegenc**	
	Medio	Máximo	Medio	Máximo	Medio	Máximo	Medio	Máximo	Medio	Máximo
C1	2,21	1,23	12,61	9,27	9,41	5,42	4,74	2,72	1,42	0,52
C2	2,16	1,21	15,27	10,79	12,17	7,01	9,34	5,32	4,87	2,42
C3	5,37	2,95	3,50	1,93	3,32	1,90	9,35	5,33	4,45	2,43

que todas las aplicaciones reducen el impacto térmico tanto en la máxima temperatura media alcanzada, como en la máxima temperatura.

Tabla 4.8: ARM-Porcentaje de mejora respecto a máxima temperatura media y máxima.

	epic		**unepic**		**cjpeg**		**djpeg**		**gsmdec**	
	Medio	Máx.	Medio	Máx.	Medio	Máx.	Medio	Máx.	Medio	Máx.
C1	3,87	2,14	5,55	3,10	1,84	1,01	1,22	0,67	4,12	2,28
C2	3,74	2,15	5,51	3,10	1,81	1,01	1,21	0,67	4,03	2,29
C3	3,78	2,15	5,54	3,11	1,83	1,01	1,22	0,67	4,13	2,29
	gsmenc		**rawcaudio**		**rawdaudio**		**mpegdec**		**mpegenc**	
	Medio	Máx.	Medio	Máx.	Medio	Máx.	Medio	Máx.	Medio	Máx.
C1	4,15	2,31	2,61	1,44	1,00	0,55	7,75	5,36	1,56	0,85
C2	4,07	2,31	2,57	1,44	1,00	0,55	7,68	5,36	1,53	0,86
C3	4,17	2,32	2,59	1,44	1,01	0,55	7,73	5,36	1,55	0,86

4.5.2.3. Análisis en porcentaje de mejora de la temperatura individual en los registros

Mirando más de cerca, es posible analizar los datos a nivel del incremento o reducción de temperatura de cada registro y en función de los accesos que el algoritmo de optimización le asigna. Con esta finalidad, se presentan un conjunto de tablas donde se muestra el porcentaje de mejora o empeoramiento que supone la solución obtenida en cuanto a cada uno de los registros y su ubicación[10]. El formato

[10]Siempre que se habla de ubicación, se hace referencia a como el MOEA redistribuye los accesos a posiciones del banco de registros que facilitan un reducción del impacto térmico.

del conjunto de tablas viene dado por los registros que componen cada arquitectura y el formato de filas y columnas correspondiente a cada configuración arquitectónica, donde los accesos a cada registro se representan en la ubicación que el MOEA ha proporcionado como solución que minimiza el impacto térmico. Las columnas de porcentajes representan la diferencia en porcentaje con respecto a la solución no optimizada, descrita al inicio de la sección. Es posible observar como el cambio de ubicación en los accesos a registros provoca que si bien hay registros que disminuyen su temperatura, hay otros que la incrementan, aunque los altos porcentajes de empeoramiento no se corresponden con un gran incremento de temperatura sino con la distribución de esta a lo largo del banco.

El formato seguido en las tablas representa en negrita aquellos valores que suponen un porcentaje de mejora significativo respecto a la solución no optimizada. Las celdas con el color del texto en rojo muestran un alto porcentaje de empeoramiento que tal y como se ha mencionado anteriormente no significa una temperatura elevada.

En primer lugar, se muestra la evolución para la arquitectura VLIW, ordenadas desde $C1$ a $C3$. La Tablas 4.9 y 4.10 representan los porcentajes respecto a la diferencia de temperatura por aplicación, para la configuración $C1$ y para cada elemento del banco de registros.

Analizando los porcentajes de mejora más altos que aparecen en las tablas 4.9 y 4.10 para la configuración $C1$, se observa como dichos valores se encuentran situados en la ubicación de los registros con mayor densidad de potencia en la distribución espacial original de los accesos (Figura 4.11). En dicha distribución, los registros globales ($R0$ a $R7$) y el punto de pila ($R17$) son los que tienen una temperatura mayor y, por tanto, la solución optimizada proporciona una distribución espacial que los aleja. El resultado permite una disminución de la temperatura no sólo por la reducción de temperatura de las celdas correspondientes a estos registros, sino también por la influencia sobre las celdas de los registros vecinos.

Valores como los que aparecen para la aplicación *unepic* de $-3236,52\,\%$ o para

Tabla 4.9: Porcentaje en diferencia de temperatura del banco de registros por aplicación. Configuración C1 (Parte 1).

Arquitectura VLIW - Configuración 1 - 32 filas *x* 1 columna - Parte 1									
epic	%	unepic	%	cjpeg	%	djpeg	%	gsmdec	%
R0	0,08	R0	0,05	R0	-0,76	R0	0,02	R0	-13,29
R1	5,90	R1	0,40	R1	-34,06	R1	0,27	R1	-96,34
R9	**51,85**	R2	1,40	R17	-87,21	R2	1,15	R11	-33,48
R3	10,23	R7	**44,13**	R15	15,34	R15	**32,58**	R7	**84,08**
R14	**69,98**	R8	**74,58**	R23	**42,71**	R5	-3,87	R2	**61,51**
R19	**53,39**	R21	**69,53**	R14	**36,19**	R23	**42,81**	R16	**77,36**
R25	**54,81**	R5	-11,15	R2	18,09	R14	**58,24**	R13	-0,11
R2	-114,23	R16	-90,06	R10	7,55	R10	0,54	R23	-13,44
R15	-8,31	R3	-20,46	R12	22,72	R6	-110,86	R6	-104,38
R5	-255,35	R13	0,26	R4	-50,47	R25	21,99	R21	**38,14**
R24	-64,70	R6	-515,87	R9	-17,51	R20	-69,42	R14	-49,28
R8	-46,20	R12	-0,05	R16	-5,35	R13	-11,97	R15	**84,87**
R20	-131,00	R11	-32,49	R18	-10,11	R19	13,51	R25	43,25
R11	-45,34	R22	19,50	R22	-5,12	R8	-1,46	R4	-63,21
R6	-244,56	R14	1,44	R13	-18,91	R21	**25,41**	R10	47,08
R23	-53,59	R25	31,18	R3	-76,84	R11	-48,76	R8	-45,65
R10	**33,92**	R9	**47,64**	R21	28,07	R18	**52,70**	R19	-8,60
R4	**54,66**	R20	**83,84**	R20	**71,53**	R4	**61,90**	R5	**22,91**
R7	**60,74**	R23	**50,64**	R19	**37,50**	R12	**31,65**	R18	14,61
R22	17,39	R4	-220,30	R24	-5,62	R3	-49,77	R9	-179,90
R21	16,26	R10	4,74	R5	-58,99	R22	-22,80	R20	-233,02
R16	33,87	R18	-73,99	R25	-14,26	R9	-44,78	R17	-1385,20
R18	-41,82	R19	-208,33	R11	-9,46	R24	20,16	R22	-453,87
R13	14,58	R15	-280,30	R8	-47,03	R16	-86,29	R12	-899,07
R12	-48,24	R24	-917,07	R7	-27,18	R7	-66,15	R24	-2000,96
R17	-594,93	R17	-3236,52	R6	-121,53	R17	-504,15	R3	-9112,64
R26	-77,52	R26	-1017,21	R26	-28,60	R26	-93,20	R26	-2255,76
R27	-1,84	R27	-22,23	R27	-0,56	R27	-1,87	R27	-43,79
R28	-0,04	R28	-0,48	R28	0,00	R28	-0,04	R28	-0,84
R29	0,00	R29	-0,01	R29	0,00	R29	0,00	R29	0,00
R30	0,00	R30	0,00	R30	0,00	R30	0,00	R30	0,00
R31	0,00	R31	0,00	R31	0,00	R31	0,00	R31	0,00

gsmdec con $-9112,64\,\%$, no implican incrementos grandes de temperatura. Concretamente, el porcentaje $-3236,52\,\%$ se encuentra en la posición que corresponde en la solución no optimizada a los accesos al registro $R25$, cuya temperatura es muy baja ($0,00587^{o}C$). En la solución optimizada se le ha asignado el valor correspondiente a los accesos al registro $R4$ cuya temperatura, en comparación, es bastante mayor ($0,19615^{o}C$). En el caso de *gsmdec*, el porcentaje $-9112,64\,\%$ que corresponde al registro $R25$ tiene un incremento de temperatura en la solución no optimizada de $0,00271^{o}C$ y en la solución optimizada se ha asignado el valor del

Tabla 4.10: Porcentaje en diferencia de temperatura del banco de registros por aplicación. Configuración C1 (Parte 2).

Arquitectura VLIW - Configuración 1 - 32 filas *x* 1 columna - Parte 2									
gsmenc	%	rawcaudio	%	rawdaudio	%	mpeg2dec	%	mpeg2enc	%
R0	-25,44	R0	-242,67	R0	-208,31	R0	-0,02	R0	-0,92
R1	-197,13	R1	-242,68	R1	-208,32	R1	-3,67	R1	-13,00
R17	-168,87	R4	-114,52	R3	-99,04	R16	-19,26	R17	-6,30
R10	**56,56**	R10	**78,23**	R15	**77,89**	R25	**31,30**	R10	**72,93**
R21	**89,38**	R18	**99,56**	R19	**99,37**	R9	-50,09	R12	**90,93**
R8	**48,46**	R21	**84,96**	R23	**83,70**	R14	**33,34**	R18	**74,63**
R19	**71,35**	R17	1,34	R17	-33,70	R3	-18,05	R15	**53,63**
R11	-221,01	R19	-4,99	R20	-77,34	R4	-42,37	R24	-278,31
R18	**34,21**	R16	**63,90**	R12	28,04	R12	-3,89	R3	-424,23
R25	**60,52**	R5	-53,60	R18	**77,57**	R6	**32,79**	R7	-110,51
R5	-427,78	R7	-61,86	R11	-293,06	R18	-29,53	R22	-51,71
R22	**57,28**	R8	9,52	R25	**76,17**	R15	**24,82**	R6	-63,58
R12	6,38	R25	**66,57**	R8	-10,59	R11	-6,41	R19	-39,24
R20	**49,24**	R2	-125,45	R13	-7,10	R24	10,62	R20	-37,02
R6	-316,17	R20	-127,94	R14	-1,81E+02	R5	-76,51	R21	-212,59
R15	-128,35	R11	-6,58E+03	R2	-1,50E+04	R10	26,51	R4	-904,55
R2	-0,16	R23	6,83	R24	23,70	R2	**37,07**	R13	**34,76**
R14	**90,12**	R6	10,26	R9	**48,37**	R23	**73,24**	R11	**92,78**
R13	**62,19**	R22	6,69	R21	**44,49**	R21	**33,73**	R9	**67,44**
R23	-99,22	R12	-1,08E+04	R6	-9,99E+03	R20	4,82	R25	-52,53
R3	-1272,21	R14	-5,31E+04	R16	-3,65E+04	R13	21,23	R8	-133,61
R24	-293,60	R9	-4,22E+05	R7	-2,26E+05	R19	-11,25	R16	-112,21
R9	-707,44	R13	-1,31E+06	R5	-4,64E+06	R22	-3,98	R5	-277,02
R16	-442,13	R15	-1,26E+07	R10	-5,64E+07	R8	-21,50	R14	-106,76
R7	-953,71	R24	-3,26E+08	R22	-1,96E+08	R7	-81,26	R23	-294,68
R4	-3175,08	R3	-3,27E+08	R4	-1,97E+08	R17	-389,46	R2	-589,03
R26	-1008,43	R26	-3,27E+08	R26	-1,97E+08	R26	-96,62	R26	-222,18
R27	-21,19	R27	-3,27E+08	R27	-1,97E+08	R27	-1,90	R27	-5,52
R28	-0,41	R28	-3,27E+08	R28	-1,97E+08	R28	-0,04	R28	-0,14
R29	-0,01	R29	-3,27E+08	R29	-1,97E+08	R29	0,00	R29	0,00
R30	0,00	R30	-3,27E+08	R30	-1,97E+08	R30	0,00	R30	0,00
R31	0,00	R31	-3,27E+08	R31	-1,97E+08	R31	0,00	R31	0,00

registro $R3$ con una temperatura de $0,24954^{o}C$, considerablemente mayor en porcentaje al valor previo. De esta forma, los accesos a $R3$ que se encuentran en la solución no optimizada entre $R2$ y $R4$, los tres con valores de incremento de temperatura comparativamente altos respecto al resto, en la solución optimizada se alejan de ellos favoreciendo la reducción térmica. Además, tal y como se puede observar los valores de incremento de temperatura en ningún caso representan incrementos significativos.

Los porcentajes en diferencia de temperatura máxima, respecto a la configuración original $C2$, se ofrecen en la Tablas 4.11, 4.12 y 4.13. La información referente a la distribución $C3$ se muestra en la Tabla 4.14. Al igual que en las tablas de la configuración $C1$, se han resaltado en negrita aquellas posiciones que al disminuir su densidad de potencia gracias a la nueva distribución espacial, permiten

reducir la temperatura del banco de registros. Los valores marcados con el texto en rojo incrementan la temperatura, aunque al igual que en la configuración anterior, este incremento comparativamente es mayor, pero no significativo. Por ejemplo, en la Tabla 4.11 para *epic* en la posición del registro $R15$ aparece un porcentaje de $-463,61\,\%$, este valor se debe a que en la solución no optimizada el incremento de temperatura es de $0,01778^{o}C$ y en la optimizada se ha situado en esa posición el registro $R3$ con un incremento de temperatura de $0,10025^{o}C$. Comparando ambos valores el incremento de temperatura en la solución optimizada es considerablemente mayor, pero no se corresponde con un valor alto de temperatura. El objetivo es alejar $R3$ de registros vecinos con incrementos de temperatura, comparativamente altos. Además, al llevar $R3$ a un extremo, en contacto directo con el exterior, se favorece la disipación de potencia y por tanto, la reducción de temperatura.

Si se observa la Tabla 4.12 para *rawdaudio*, el valor resaltado en rojo con un porcentaje de $-62467,77\,\%$ llama considerablemente la atención, sin embargo si se analizan los datos se concluye que el incremento de temperatura no es significativo, aunque dicha posición se ve directamente influenciada por la temperatura de sus vecinos. El empeoramiento se debe principalmente a que dicha posición en la solución no optimizada tenía un incremento de temperatura de $0,00004469^{o}C$ y en la solución optimizada de $0,02795^{o}C$, y si bien ambos valores no son representativos, la diferencia entre ellos lleva a un porcentaje de empeoramiento muy elevado. La explicación se encuentra en los accesos al registro situado como vecino que en la solución no optimizada son los correspondientes a $R15$ con un incremento de temperatura de $0,02378^{o}C$ y en la solución optimizada se sitúan los de $R4$ con un incremento de temperatura de $0,1432^{o}C$.

En cuanto a ARM, se realiza también la extensión de este análisis para cada aplicación, configuración y distribución espacial de los accesos a los registros respecto a la distribución espacial original. De esta forma, en las tablas 4.15 y 4.16 se muestran los porcentajes diferencia en cuanto a la distribución de la temperatura para la configuración $C1$ para la arquitectura ARM. Se resaltan con el color del

Tabla 4.11: Porcentaje en diferencia de temperatura del banco de registros por aplicación. Configuración C2 (Parte 1)

Arquitectura VLIW - Configuración 2 - 16 filas *x* 2 columnas (Parte 1).							
epic				**unepic**			
Col.1	%	%	Col.2	Col.1	%	%	Col.2
R0	-15,74	-184,88	R17	R0	-110,50	-220,86	R17
R1	**20,78**	**69,87**	R10	R1	**57,21**	**75,68**	R7
R16	**58,82**	**73,02**	R11	R14	**37,85**	**59,48**	R21
R9	**71,94**	**31,41**	R21	R23	**60,81**	**29,91**	R11
R5	**29,37**	**24,70**	R22	R8	**71,51**	**27,50**	R19
R13	**61,79**	**36,44**	R24	R3	**55,03**	**27,81**	R25
R25	**52,98**	-25,27	R2	R15	**49,36**	-209,13	R5
R20	-44,39	-19,56	R8	R12	-78,28	-169,57	R22
R23	-17,97	-43,38	R7	R6	-73,47	-219,24	R24
R19	-127,54	-242,41	R4	R16	-33,36	-1007,55	R4
R15	-59,28	-63,30	R26	R9	-219,11	-448,24	R26
R6	-231,69	-46,34	R27	R13	17,65	4,77	R27
R14	-63,71	-14,89	R28	R20	-35,31	-18,99	R28
R18	-177,38	-31,90	R29	R18	**30,33**	**30,59**	R29
R12	-108,40	-25,38	R30	R10	**61,17**	**43,25**	R30
R3	-463,61	-86,06	R31	R2	-284,94	-169,04	R31
cjpeg				**djpeg**			
Col.1	%	%	Col.2	Col.1	%	%	Col.2
R0	-23,78	-115,36	R17	R0	-26,62	-64,22	R2
R1	24,91	**56,53**	R15	R1	**32,28**	**63,76**	R7
R2	3,63	30,41	R23	R6	21,51	9,89	R9
R12	**35,99**	14,29	R20	R10	**46,08**	5,37	R21
R24	**35,26**	5,35	R11	R14	4,25	-22,51	R3
R5	11,74	-26,83	R8	R19	**47,81**	-9,42	R11
R14	**50,01**	12,59	R9	R22	**64,83**	28,57	R23
R25	19,775	7,28	R21	R20	-5,94	-35,88	R5
R19	27,39	-3,26	R10	R13	8,25	-20,43	R24
R22	-4,49	-60,71	R4	R15	37,70	-161,39	R16
R18	-13,69	-15,28	R26	R8	-123,30	-48,89	R26
R16	-5,35	-1,58	R27	R12	9,07	0,53	R27
R13	-17,12	-4,77	R28	R18	0,18	1,39	R28
R3	-77,34	-15,40	R29	R4	10,32	4,72	R29
R7	-28,98	-7,56	R30	R25	16,30	1,48	R30
R6	-124,91	-21,82	R31	R17	-371,32	-84,42	R31

texto en rojo alguno de los valores que debido a la distribución optimizada incrementan su temperatura y en negrita aquellas posiciones que reducen su temperatura. La reducción del impacto térmico se debe al alejamiento de aquellos registros con mayor incremento de temperatura o situándolos en posiciones externas para una mejor disipación del calor.

La tablas 4.17 y 4.18 permiten analizar el impacto de la optimización para las configuraciones $C2$ y $C3$, respectivamente. En todas las tablas se observa como en las soluciones aportadas por el proceso de optimización se distribuyen los accesos

Tabla 4.12: Porcentaje en diferencia de temperatura del banco de registros por aplicación. Configuración C2 (Parte 2)

Arquitectura VLIW - Configuración 2 - 16 filas *x* 2 columnas (Parte 2).							
gsmdecode				**gsmencode**			
Col.1	%	%	Col.2	Col.1	%	%	Col.2
R0	-431,44	-803,19	R3	R0	14,50	-9,01	R5
R1	21,01	**44,04**	R7	R1	**52,23**	**90,25**	R15
R17	14,87	21,49	R15	R3	-20,53	5,25	R2
R23	**89,24**	**40,92**	R12	R25	**77,82**	**32,13**	R23
R2	**61,55**	6,99	R9	R13	**78,69**	-37,73	R6
R10	**79,01**	14,32	R22	R16	**73,15**	**31,52**	R19
R6	7,30	-293,88	R13	R11	24,52	-82,94	R18
R20	-20,16	-211,91	R21	R14	-28,62	-67,31	R22
R18	24,89	-267,13	R16	R12	0,34	-266,76	R7
R14	-95,46	-1376,26	R4	R24	7,91	-1367,14	R17
R24	**43,46**	-198,33	R26	R8	-267,40	-734,85	R26
R5	**51,52**	**48,34**	R27	R21	**66,74**	**45,37**	R27
R19	**57,24**	**50,15**	R28	R9	17,35	16,94	R28
R8	**61,23**	**56,25**	R29	R20	**55,38**	**38,99**	R29
R25	**44,54**	27,97	R30	R10	-187,37	-124,58	R30
R11	-535,97	-319,72	R31	R4	-2840,64	-803,20	R31
rawcaudio				**rawdaudio**			
Col.1	%	%	Col.2	Col.1	%	%	Col.2
R0	-794,93	-1181,43	R4	R0	-603,71	-902,18	R3
R1	**31,71**	28,90	R24	R1	21,85	**44,40**	R22
R5	**34,63**	**45,119**	R10	R17	16,72	**34,93**	R14
R18	**93,07**	**80,59**	R21	R24	**90,48**	**76,66**	R15
R11	**83,65**	21,95	R9	R12	**76,54**	-28,80	R9
R7	**85,94**	**61,87**	R15	R25	**87,64**	**53,46**	R21
R6	11,51	-86,07	R13	R2	2,51	-233,27	R8
R19	5,68	-70,11	R23	R19	-26,12	-255,98	R18
R14	**43,07**	-439,71	R25	R16	**70,61**	-143,36	R10
R2	-105,93	-2209,42	R3	R6	-16,05	-940,71	R5
R22	-137,41	-1663,99	R26	R7	-30,94	-695,97	R26
R8	9,59	12,48	R27	R11	5,481	8,38	R27
R16	**77,35**	**70,28**	R28	R20	**74,01**	**66,74**	R28
R12	-48,41	-33,64	R29	R13	19,32	25,63	R29
R20	-169,52	-190,13	R30	R23	-443,79	-438,68	R30
R17	-1,23E+04	-8556,69	R31	R4	-1,18E+05	-62467,77	R31

a registros a lo largo de la estructura, de tal manera que los puntos más calientes se encuentren lo más alejados posibles. Al igual que sucedía con VLIW, los porcentajes negativos se refieren a un incremento térmico respecto a la distribución no optimizada no sólo a consecuencia de los accesos al propio registro sino también de la influencia de los registros vecinos.

Tabla 4.13: Porcentaje en diferencia de temperatura del banco de registros por aplicación. Configuración C2 (Parte 3)

Arquitectura VLIW - Configuración 2 - 16 filas *x* 2 columnas (Parte 3).							
mpeg2decode				**mpeg2encode**			
Col.1	%	%	Col.2	Col.1	%	%	Col.2
R0	7,97	22,51	R16	R0	-91,57	-230,05	R17
R1	22,53	**58,43**	R3	R1	**32,84**	**74,67**	R10
R20	17,67	**36,56**	R4	R2	9,44	**44,33**	R13
R12	11,56	-3,39	R11	R7	**76,61**	**59,96**	R20
R14	19,46	**33,88**	R7	R12	**89,61**	**43,51**	R25
R19	11,92	-20,23	R2	R9	**72,72**	18,20	R16
R23	19,43	19,09	R24	R18	**37,53**	-28,01	R19
R18	-55,77	-65,71	R5	R14	-52,47	-38,37	R15
R13	17,82	-15,37	R25	R5	-14,83	-153,56	R21
R21	26,37	-79,44	R9	R22	-76,11	-445,00	R4
R8	-20,52	-24,15	R26	R8	-243,46	-167,80	R26
R6	**24,35**	7,87	R27	R11	-34,40	-15,81	R27
R15	**24,97**	8,60	R28	R6	-60,41	-14,34	R28
R22	-5,77	-0,68	R29	R23	-11,30	-5,14	R29
R10	-63,33	-20,55	R30	R24	-267,42	-125,59	R30
R17	-285,76	-67,66	R31	R30	-1349,26	-432,16	R31

4.5.2.4. Análisis de la potencia individual consumida por los registros

Para finalizar, se analizan los datos desde el punto de vista de la potencia consumida para cada posición del banco de registros y cada modelo arquitectónico. Se presenta un resumen con la máxima potencia obtenida por la posición donde el MOEA asigna los accesos a cada registro y su porcentaje diferencia con la ubicación original. Las tablas 4.19, 4.20, 4.21, 4.22, 4.23 y 4.23 se refieren a los valores obtenidos para las aplicaciones sobre una arquitectura VLIW para las configuraciones $C1$, $C2$ y $C3$, respectivamente. Para todas las tablas las columnas identificadas como $N - Opt$ representan la distribución no optimizada y las columnas sombreadas en azul e identificadas con Opt las soluciones optimizadas. Para claridad en la interpretación de los resultados, algunos de los valores son marcados con el fin de identificar cómo el algoritmo de optimización, tal y como se esperaba, sitúa los accesos a los registros con un consumo mayor de potencia alejados entre sí, con el fin de reducir el impacto térmico. De esta forma, se marcan en color naranja celdas de

Tabla 4.14: Porcentaje en diferencia de temperatura del banco de registros por aplicación. Configuración C3.

Arquitectura VLIW - Configuración 3 - 8 filas *x* 4 columna (Parte 1)							
epic							
Col.1	%	Col.2	%	Col.3	%	Col.4	%
R0	0,04	R2	-61,95	R19	**38,86**	R17	-575,52
R1	3,47	R13	8,47	R25	**85,07**	R10	-101,95
R14	**61,70**	R7	-9,36	R9	**65,75**	R26	8,68
R8	**74,27**	R5	-226,73	R6	-56,29	R27	-19,42
R3	-10,84	R22	-26,07	R20	-40,83	R28	-161,52
R12	**45,87**	R11	**54,60**	R23	**45,96**	R29	-41,08
R16	**63,39**	R15	1,09	R24	**38,17**	R30	1,06
R21	2,98	R4	-260,83	R18	-73,16	R31	0,04
unepic							
Col.1	%	Col.2	%	Col.3	%	Col.4	%
R0	**66,52**	R9	**41,28**	R19	17,41	R17	-1951,72
R1	**84,17**	R15	**45,33**	R12	**86,47**	R10	-463,50
R7	**75,61**	R22	0,72	R25	**67,48**	R26	-37,81
R2	-125,11	R6	-126,54	R5	-83,91	R27	-101,52
R20	**77,11**	R3	-25,35	R14	-225,01	R28	-1011,60
R21	**79,78**	R16	**64,48**	R24	**34,02**	R29	-302,37
R23	**57,32**	R18	**71,62**	R13	6,20	R30	-4,42
R4	-347,31	R8	4,57	R11	-188,01	R31	-0,07
cjpeg							
Col.1	%	Col.2	%	Col.3	%	Col.4	%
R0	12,59	R22	25,27	R18	29,03	R6	-117,67
R1	24,97	R21	14,99	R25	**70,94**	R15	-24,50
R7	20,35	R9	6,68	R24	**34,04**	R26	-3,15
R17	-90,55	R4	-53,72	R8	-37,17	R27	-12,50
R20	**41,22**	R5	-35,93	R19	-4,77	R28	-21,05
R23	**45,71**	R13	24,83	R11	29,35	R29	-6,74
R10	**43,65**	R12	11,01	R14	3,42	R30	0,04
R2	-56.43	R16	-14.40	R3	-65.14	R31	0.00
djpeg							
Col.1	%	Col.2	%	Col.3	%	Col.4	%
R0	-7,90	R22	18,80	R9	**43,22**	R5	-125,72
R1	**61,01**	R19	**38,26**	R21	**69,93**	R3	-112,45
R10	**69,30**	R7	4,99	R20	**30,16**	R26	-7,53
R2	-98,92	R6	-164,27	R11	-79,97	R27	-54,55
R16	-32,39	R4	8,26	R8	-7,30	R28	-35,14
R24	**53,70**	R23	**42,09**	R25	**41,17**	R29	-32,88
R12	**65,31**	R18	22,56	R15	6,97	R30	-7,75
R14	-45.70	R13	-121.58	R17	-223.13	R31	-0.23
gsmdecode							
Col.1	%	Col.2	%	Col.3	%	Col.4	%
R0	**72,90**	R6	-28,00	R14	-92,62	R11	-4130,53
R1	**80,87**	R22	**59,32**	R19	**77,23**	R10	-1012,70
R7	**51,83**	R23	**50,90**	R21	35,09	R26	-294,90
R3	1,84	R2	**50,91**	R5	-460,04	R27	-1103,87
R8	**71,77**	R9	8,36	R12	-329,81	R28	-1235,18
R18	**79,91**	R15	**78,27**	R20	17,64	R29	-398,70
R24	**63,58**	R25	**63,89**	R16	-198,45	R30	-9,84
R13	-94,21	R17	-119,80	R4	-2690,33	R31	-0,21
gsmencode							
Col.1	%	Col.2	%	Col.3	%	Col.4	%
R0	36,07	R11	24,37	R13	46,18	R4	-3,50E+03
R1	**71,25**	R19	**70,69**	R16	**89,83**	R7	-631,10
R10	34,77	R14	21,73	R20	**75,52**	R26	-95,73
R17	-114,48	R12	-18,88	R6	-318,79	R27	-380,04
R3	36,99	R2	-8,50	R9	-205,04	R28	-905,10
R24	**74,78**	R15	**72,84**	R23	**57,33**	R29	-262,77
R21	**63,68**	R22	39,82	R25	11,58	R30	-3,15
R18	-112,13	R8	-316,75	R5	-915,02	R31	-0,04
Col.1	%	Col.2	%	Col.3	%	Col.4	%
R0	**70,52**	R8	**41,12**	R9	-123,55	R4	-1,10E+07
R1	**84,56**	R22	**81,27**	R19	**84,38**	R25	-2,55E+06
R10	**51,72**	R24	**38,33**	R7	11,03	R26	-4,49E+05
R3	1,78	R2	-51,93	R6	-15860,71	R27	-5,23E+07
R5	**70,13**	R17	-74,98	R11	-1515,26	R28	-1,04E+09
R21	**85,91**	R23	**48,90**	R18	-44,12	R29	-1,08E+08
R15	**77,29**	R16	**52,69**	R20	-136,95	R30	-8,91E+06
R13	-12,22	R14	-79,64	R12	-34009,88	R31	-7,14E+06
mpegdecode							
Col.1	%	Col.2	%	Col.3	%	Col.4	%
R0	**70,92**	R6	-28,00	R14	-92,62	R11	-4130,53
R1	**80,87**	R22	**59,32**	R19	**77,23**	R10	-1012,70
R7	**51,83**	R23	**50,90**	R21	35,09	R26	-294,90
R3	1,84	R2	**50,91**	R5	-460,04	R27	-1103,87
R8	**71,77**	R9	8,36	R12	-329,81	R28	-1235,18
R18	**79,91**	R15	**78,27**	R20	17,64	R29	-398,70
R24	**63,58**	R25	**63,89**	R16	-198,45	R30	-9,84
R13	-94,21	R17	-119,80	R4	-2690,33	R31	-0,21
mpegencode							
Col.1	%	Col.2	%	Col.3	%	Col.4	%
R0	1,89	R22	4,09	R25	23,48	R4	-397,74
R1	**63,46**	R13	16,08	R11	**88,58**	R23	-10,33
R24	**72,85**	R10	-58,21	R18	**69,33**	R26	12,77
R2	-5,26	R17	-98,04	R6	-97,48	R27	-0,69
R5	1,33	R8	-564,91	R3	-572,86	R28	-40,14
R20	**39,58**	R16	4,71	R7	7,42	R29	2,61
R15	**61,34**	R9	**49,70**	R21	**46,29**	R30	6,38
R12	-5,25	R19	-19,69	R14	-57,98	R31	0,20

la tabla que representan los accesos a registros de la distribución no optimizada, los cuales rodean a registros con un consumo de potencia alto. En la solución optimizada se puede observar que han sido sustituidos por otros con un consumo menor de potencia, lo cuales se marcan en color azul.

Por ejemplo, en la Tabla 4.19 para la aplicación *epic*, en la solución optimizada en la cuarta posición aparecen los accesos a $R3$ entre los accesos a los registros $R9$ y $R14$ en lugar de $R2$ y $R4$ que aparecían en la distribución no optimizada, esto es debido a que $R9$ y $R14$ tienen un menor consumo de potencia y por tanto,

Tabla 4.15: Porcentaje en diferencia de temperatura del banco de registros por aplicación

Arquitectura ARM - Configuración 1 - 16 filas *x* 1 columna - Parte 1									
R.	epic	R.	unepic	R.	cjpeg	R.	djpeg	R.	gsmdec
R0	2,14	R0	3,10	R0	1,01	R0	0,67	R0	2,28
R11	**56,37**	R11	**65,32**	R12	**37,32**	R11	**28,30**	R9	**57,17**
R9	**82,98**	R7	**81,30**	R11	**85,82**	R12	**85,31**	R5	**80,73**
R8	**63,97**	R8	**59,37**	R8	**50,46**	R9	**66,68**	R12	**82,36**
R3	**72,82**	R5	**38,40**	R2	**34,64**	R10	**50,50**	R7	**47,84**
R10	**47,61**	R4	-36,15	R6	15,23	R7	25,06	R1	-328,70
R1	-348,20	R10	-17,79	R5	-4,52	R5	15,96	R8	-109,40
R7	-127,50	R2	-296,29	R3	-60,40	R6	-49,24	R4	-223,43
R5	-50,88	R9	13,29	R4	-304,65	R3	-127,59	R6	-299,58
R2	-393,98	R3	-382,35	R7	9,56	R2	-263,52	R2	-2317,34
R6	-269,51	R6	-164,73	R10	-1,24	R8	-169,98	R11	-421,25
R12	-2115,03	R12	-1105,97	R9	-92,20	R1	-2011,07	R10	-42,48
R4	-3422,26	R1	-1906,51	R1	-869,15	R4	-130,17	R3	-426,08
R13	-197,24	R13	-136,10	R13	-50,91	R13	-2,14	R13	-112,48
R14	-10,38	R14	-7,53	R14	-3,30	R14	-0,57	R14	-2,84
R15	-0,06	R15	-0,09	R15	-0,13	R15	-0,04	R15	-0,06

Tabla 4.16: Porcentaje en diferencia de temperatura del banco de registros por aplicación

Arquitectura ARM - Configuración 1 - 16 filas *x* 1 columna - Parte 2									
R.	gsmenc	R.	rawcaudio	R.	rawdaudio	R.	mpeg2dec	R.	mpeg2enc
R0	2,31	R0	1,44	R0	-10,51	R0	5,36	R0	0,85
R9	**57,47**	R11	**46,36**	R11	-15,45	R9	**72,13**	R12	**33,43**
R11	**78,10**	R12	**93,35**	R12	**94,01**	R6	**70,83**	R3	25,31
R12	**82,06**	R8	-55,29	R7	-64,67	R5	-9,45	R5	-17,00
R5	**44,43**	R4	-0,41	R6	24,90	R4	1,97	R7	**46,34**
R2	-269,70	R5	20,37	R4	-59,07	R7	-7,84	R6	20,49
R7	-107,72	R7	**51,23**	R5	-18,75	R8	-407,50	R1	-54,31
R4	-230,84	R9	26,43	R8	-49,75	R11	-5,35	R2	23,06
R8	-302,69	R2	-149,13	R1	-2,89	R2	68,40	R4	-69,04
R10	-3052,79	R10	-51,69	R9	-2,94	R3	8,89	R8	**85,89**
R6	-365,84	R6	-211,27	R10	-7,62	R10	-874,84	R10	**29,27**
R10	-44,58	R3	-372,45	R3	-433,69	R1	-2047,96	R11	-121,19
R3	-431,02	R1	-1050,96	R2	-2366,85	R12	-300,19	R9	-3922,36
R13	-114,73	R13	-49,30	R13	-63,03	R13	-0,59	R13	-275,72
R14	-2,86	R14	-5,06	R14	-133,89	R14	-0,10	R14	-15,09
R15	-0,06	R15	-2,09	R15	-36,14	R15	-0,00	R15	-0,62

facilitan la reducción de temperatura del banco de registros, haciendo de sumideros de temperatura.

Tabla 4.17: Porcentaje en diferencia de temperatura del banco de registros por aplicación

Arquitectura ARM - Configuración 2 - 8 filas *x* 2 columnas							
epic				unepic			
R.	Col. 1	Col. 2	R.	R.	Col. 1	Col. 2	R.
R4	**34.51**	-281.77	R0	R2	**60.90**	-250.80	R0
R12	**69.16**	-46.19	R11	R7	**78.99**	-26.03	R11
R5	**73.43**	**53.17**	R10	R4	**56.16**	**48.80**	R9
R8	**69.70**	-32.35	R9	R6	**62.92**	-42.07	R10
R6	**80.41**	-92.33	R2	R8	27.36	-367.66	R3
R3	17.71	-22.62	R13	R5	5.76	-43.14	R13
R7	-47.01	-13.36	R14	R12	-49.27	-15.95	R14
R1	-620.91	-206.80	R15	R1	-743.42	-280.73	R15
cjpeg				djpeg			
R.	Col. 1	Col. 2	R.	R.	Col. 1	Col. 2	R.
R10	**69.95**	-360.40	R0	R2	**76.45**	-376.78	R0
R6	**64.54**	-89.48	R12	R6	**69.65**	-154.33	R11
R2	24.31	**78.83**	R11	R7	**60.02**	**49.35**	R12
R5	-13.05	11.10	R8	R3	11.96	-44.15	R9
R9	**46.90**	-62.65	R3	R4	0.40	-19.85	R10
R4	-29.22	-11.97	R13	R5	2.05	-0.59	R13
R7	11.01	7.84	R14	R8	9.78	15.01	R14
R1	-272.15	-121.14	R15	R1	-272.03	-137.76	R15
gsmdec				gsmenc			
R.	Col. 1	Col. 2	R.	R.	Col. 1	Col. 2	R.
R1	**51.42**	-353.27	R0	R1	**51.23**	-354.07	R0
R6	**72.03**	-75.84	R9	R6	72.169	-75.62	R9
R12	**76.11**	**52.93**	R7	R11	**75.41**	**54.44**	R7
R11	**80.79**	**52.49**	R10	R4	**73.68**	**49.46**	R10
R4	25.03	-146.85	R2	R5	47.17	-150.21	R2
R5	-12.35	-58.43	R13	R12	-19.14	-60.46	R13
R8	-146.98	-28.51	R14	R8	-151.54	-28.95	R14
R3	-973.65	-128.26	R15	R3	-1002.49	-130.06	R15
rawcaudio				rawdaudio			
R.	Col. 1	Col. 2	R.	R.	Col. 1	Col. 2	R.
R1	**62.68**	-310.90	R0	R10	**69.01**	-1048.37	R0
R3	**75.91**	-58.25	R11	R4	26.74	-141.13	R11
R2	17.56	**55.34**	R12	R9	**56.18**	**76.59**	R12
R9	-102.26	-216.11	R10	R6	-88.26	-177.389	R3
R4	-2.44	-160.06	R5	R8	2.38	-835.65	R1
R7	14.86	-4.40	R13	R5	-13.17	-19.48	R13
R8	**52.80**	**39.20**	R14	R7	-69.50	-205.53	R14
R6	-56.03	-38.22	R15	R2	-340.06	-5090.76	R15
mpeg2dec				mpeg2enc			
R.	Col. 1	Col. 2	R.	R.	Col. 1	Col. 2	R.
R1	25.66	-81.87	R0	R4	**62.85**	-261.91	R0
R10	**77.31**	8.74	R9	R2	**62.65**	**53.15**	R12
R11	**82.44**	**48.06**	R7	R1	-20.02	**83.26**	R8
R5	4.20	-94.00	R6	R6	-16.40	-51.60	R3
R3	26.59	-140.30	R4	R7	**39.07**	-304.68	R10
R2	-19.37	-9.02	R13	R5	20.39	-48.14	R13
R12	-35.81	4.29	R14	R11	-27.05	-24.63	R14
R8	-958.46	-189.57	R15	R9	-315.57	-241.96	R15

Tabla 4.18: Porcentaje en diferencia de temperatura del banco de registros por aplicación

Arquitectura ARM - Configuración 3 - 2 filas *x* 8 columna							
epic							
R4	R10	R2	R6	R3	R11	R8	R14
42.34	**55.61**	**60.02**	39.90	-144.26	-576.72	-644.66	-72.94
69.80	-25.34	37.23	5.17	-369.46	-6324.88	-287.56	-8.00
R12	R1	R7	R5	R9	R0	R13	R15
unepic							
R11	R5	R3	R9	R4	R12	R8	R14
77.04	67.22	-30.65	-18.57	-49.15	-136.39	-342.17	-79.12
-47.36	27.89	19.87	-266.06	-189.77	-2286.76	-112.22	-11.35
R0	R6	R10	R2	R7	R1	R13	R15
cjpeg							
R12	R6	R2	R7	R10	R11	R5	R14
76.24	**53.73**	36.07	16.50	-194.49	54.88	-116.77	-91.18
-162.26	-66.15	23.63	-112.15	-52.28	-308.54	-40.35	-16.60
R0	R9	R3	R4	R8	R1	R13	R15
djpeg							
R0	R8	R7	R10	R5	R1	R9	R14
0.66	12.80	21.08	38.07	-125.07	-599.28	-75.17	-11.54
28.05	**23.68**	-20.47	-71.12	-118.73	-117.15	-0.52	-0.64
R11	R4	R6	R2	R3	R12	R13	R15
gsmdec							
R9	R11	R2	R6	R1	R10	R12	R14
76.15	**73.71**	-67.13	-89.37	-1303.03	-299.75	-15.07	0.00
-76.89	52.99	38.50	-156.12	-591.85	-528.34	-88.25	0.00
R0	R5	R7	R4	R8	R3	R13	R15
gsmenc							
R0	R11	R10	R4	R7	R1	R12	R14
2.27	48.43	32.96	-117.86	-450.46	-871.22	-57.97	0.00
57.18	72.05	-192.47	-169.82	-1297.39	-75.27	-12.02	0.00
R9	R5	R3	R6	R2	R8	R13	R15
rawcaudio							
R0	R9	R7	R3	R2	R12	R5	R14
1.41	26.33	18.04	**65.75**	-99.49	-32.28	-259.83	-215.82
45.92	1.63	11.72	-53.95	-45.34	-871.16	-36.01	-13.84
R11	R4	R8	R6	R10	R1	R13	R15
rawdaudio							
R11	R10	R9	R5	R2	R7	R1	R14
76.47	**64.52**	27.87	-38.01	-145.83	7.26	-187.86	-234.11
-223.28	-68.98	0.66	-31.17	36.57	-166.62	-10.21	-16.12
R0	R3	R6	R8	R12	R4	R13	R15
mpegdec							
R0	R7	R2	R12	R3	R1	R11	R14
5.23	**30.42**	9.91	19.06	50.10	-680.25	-275.92	-6.99
72.11	35.90	-26.21	-612.54	-45.90	-241.19	-9.54	-0.61
R9	R4	R6	R8	R5	R10	R13	R15
mpegenc							
R12	R7	R6	R1	R8	R9	R10	R14
76.27	**35.27**	**57.40**	-20.80	-1.14	-138.04	-488.74	-448.36
-178.34	-90.46	**26.34**	**51.88**	**54.79**	-3.00	-124.65	-96.12
R0	R2	R5	R3	R4	R11	R13	R15

Tabla 4.19: VLIW-Potencia por registro y aplicación. Configuración C1 (parte 1).

	epic		unepic		cjpeg		djpeg		gsmdec	
	N-Opt	Opt	N-Opt	Opt	N-Opt	Opt	N-Opt	Opt	N-Opt	Opt
1	9,97E-05	9,97E-05	5,39E-06	5,39E-06	5,58E-05	5,58E-05	1,45E-05	1,45E-05	2,33E-06	2,33E-06
2	5,15E-05	5,15E-05	3,95E-06	3,95E-06	5,21E-05	5,21E-05	1,43E-05	1,43E-05	2,28E-06	2,28E-06
3	5,16E-05	1,58E-05	0,0002	0,0002	0,0001	0,0002	0,0002	0,0002	7,59E-05	0,0002
4	0,0001	0,0001	3,99E-05	2,90E-06	9,78E-05	5,09E-05	4,64E-05	1,75E-05	0,0002	2,06E-06
5	8,45E-05	1,57E-05	0,0001	3,26E-05	9,22E-05	5,34E-05	4,67E-05	5,32E-05	0,0002	7,59E-05
6	6,77E-05	3,48E-05	7,52E-05	1,27E-05	9,11E-05	5,18E-05	5,32E-05	3,43E-05	8,11E-05	2,64E-06
7	6,25E-05	2,28E-05	5,87E-05	7,52E-05	0,0001	0,0001	0,0001	4,93E-05	8,28E-05	9,15E-05
8	1,63E-05	5,16E-05	2,90E-06	1,64E-05	5,09E-05	5,10E-05	1,41E-05	1,41E-05	2,06E-06	2,67E-06
9	2,37E-05	1,62E-05	3,26E-05	3,99E-05	8,20E-05	5,26E-05	5,31E-05	0,0001	3,67E-05	8,28E-05
10	1,58E-05	6,77E-05	2,84E-05	2,05E-05	5,21E-05	9,22E-05	5,11E-05	1,80E-05	3,76E-05	6,03E-06
11	1,51E-05	2,08E-05	2,90E-06	5,87E-05	5,10E-05	5,21E-05	1,41E-05	3,51E-05	2,05E-06	6,08E-05
12	1,75E-05	2,37E-05	2,93E-05	1,88E-05	5,30E-05	5,51E-05	4,53E-05	5,03E-05	0,0002	1,29E-05
13	1,54E-05	3,97E-05	1,88E-05	2,93E-05	5,26E-05	5,84E-05	3,56E-05	2,81E-05	3,32E-05	2,37E-06
14	1,65E-05	1,75E-05	2,05E-05	1,07E-05	5,22E-05	5,40E-05	5,03E-05	5,31E-05	9,15E-05	0,0002
15	1,57E-05	6,25E-05	0,0001	0,0001	5,18E-05	5,22E-05	4,93E-05	3,20E-05	6,08E-05	2,05E-06
16	1,62E-05	2,25E-05	2,12E-05	5,41E-06	5,09E-05	9,78E-05	1,75E-05	4,53E-05	1,29E-05	3,67E-05
17	1,84E-05	1,51E-05	1,64E-05	2,84E-05	5,51E-05	5,33E-05	7,17E-05	3,68E-05	2,64E-06	1,17E-05
18	0,0002	8,45E-05	0,0002	2,90E-05	0,0002	5,41E-05	0,0001	4,67E-05	1,063E-04	8,11E-05
19	5,00E-05	1,63E-05	2,66E-05	5,15E-06	5,84E-05	5,63E-05	3,68E-05	3,56E-05	1,38E-05	1,38E-05
20	3,48E-05	3,16E-05	3,57E-05	0,0001	5,63E-05	5,24E-05	2,81E-05	4,64E-05	1,17E-05	3,76E-05
21	3,97E-05	3,41E-05	2,90E-05	2,90E-06	5,41E-05	9,11E-05	3,51E-05	4,01E-05	7,79E-06	7,79E-06
22	3,41E-05	1,84E-05	1,27E-05	2,66E-05	5,33E-05	5,32E-05	3,20E-05	5,11E-05	6,03E-06	1,06E-04
23	3,16E-05	5,00E-05	1,07E-05	3,57E-05	5,40E-05	5,30E-05	4,01E-05	2,03E-05	4,34E-06	4,34E-06
24	2,25E-05	1,65E-05	5,15E-06	2,12E-05	5,34E-05	8,20E-05	3,43E-05	7,17E-05	2,67E-06	3,32E-05
25	2,08E-05	1,54E-05	4,03E-06	4,03E-06	5,24E-05	5,09E-05	2,03E-05	1,41E-05	2,55E-06	2,55E-06
26	2,28E-05	0,0002	5,41E-06	0,0002	5,32E-05	0,0001	1,80E-05	0,0001	2,37E-06	0,0002
27	1,96E-05	1,96E-05	3,48E-06	3,48E-06	5,27E-05	5,27E-05	2,33E-05	2,33E-05	2,24E-06	2,24E-06
28	1,52E-05	1,52E-05	3,40E-06	3,40E-06	5,18E-05	5,18E-05	2,36E-05	2,36E-05	2,25E-06	2,25E-06
29	1,52E-05	1,52E-05	2,98E-06	2,98E-06	5,17E-05	5,17E-05	1,86E-05	1,86E-05	2,24E-06	2,24E-06
30	1,80E-05	1,80E-05	2,92E-06	2,92E-06	5,19E-05	5,19E-05	1,77E-05	1,77E-05	2,25E-06	2,25E-06
31	1,65E-05	1,65E-05	2,92E-06	2,92E-06	5,16E-05	5,16E-05	1,77E-05	1,77E-05	2,24E-06	2,24E-06
32	1,59E-05	1,59E-05	3,46E-06	3,46E-06	5,15E-05	5,15E-05	1,75E-05	1,75E-05	2,25E-06	2,25E-06

Tabla 4.20: VLIW-Potencia por registro y aplicación. Configuración C1 (parte 2).

	gsmencode		rawcaudio		rawdaudio		mpegdecode		mpegencode	
	N-Opt	Opt	N-Opt	Opt	N-Opt	Opt	N-Opt	Opt	N-Opt	Opt
1	4,18E-06	4,18E-06	0,000	0,0000	0,00E+00	0,00E+00	3,24E-05	3,24E-05	1,08E-05	1,08E-05
2	4,12E-06	4,12E-06	7,41E-11	7,41E-11	7,41E-11	7,41E-11	2,05E-05	2,05E-05	7,23E-06	7,23E-06
3	1,29E-04	4,27E-04	9,14E-06	2,53E-04	7,40E-05	2,43E-04	2,89E-05	3,59E-05	1,56E-04	1,83E-04
4	1,91E-04	3,83E-06	2,42E-4	0,00E+00	2,43E-04	0,00E+00	2,67E-05	1,36E-05	1,55E-04	4,70E-06
5	2,94E-04	1,13E-05	2,53E-04	6,50E-08	1,46E-04	6,37E-08	1,80E-05	3,27E-05	1,23E-04	1,12E-05
6	1,44E-04	9,98E-05	7,71E-05	9,78E-09	1,26E-04	0,00E+00	2,88E-05	1,45E-05	3,29E-05	1,35E-05
7	1,03E-04	1,24E-05	7,42E-05	8,28E-05	5,27E-05	1,02E-04	1,99E-05	2,67E-05	1,89E-05	9,35E-06
8	3,83E-06	9,09E-05	0,00E+00	3,82E-08	0,00E+00	1,14E-08	1,22E-05	1,80E-05	4,64E-06	6,40E-06
9	9,98E-05	5,79E-05	4,54E-05	9,61E-08	5,23E-05	4,14E-05	1,99E-05	2,38E-05	2,85E-05	1,55E-04
10	9,49E-05	8,30E-06	4,54E-05	7,71E-05	5,23E-05	4,42E-08	3,27E-05	1,99E-05	1,43E-05	4,64E-06
11	3,83E-06	1,44E-04	0,00E+00	0,00E+00	0,00E+00	5,23E-05	1,22E-05	2,20E-05	4,70E-06	8,20E-06
12	9,09E-05	1,06E-05	4,54E-05	4,54E-05	5,23E-05	7,41E-11	2,72E-05	1,77E-05	1,11E-05	1,89E-05
13	1,08E-04	1,08E-04	4,54E-05	7,41E-11	4,14E-05	5,23E-05	2,38E-05	2,72E-05	1,12E-05	1,47E-05
14	5,52E-05	1,21E-05	2,28E-05	6,91E-05	2,62E-05	2,62E-05	1,74E-05	1,33E-05	7,80E-06	1,09E-05
15	1,32E-05	1,03E-04	2,18E-06	9,85E-09	2,22E-10	2,22E-10	1,45E-05	2,88E-05	1,03E-05	7,67E-06
16	7,49E-06	7,49E-06	4,55E-08	4,54E-05	0,00E+00	7,40E-05	1,77E-05	1,22E-05	9,35E-06	1,23E-04
17	3,51E-05	1,29E-04	9,61E-08	0,00E+00	1,11E-07	0,00E+00	3,59E-05	2,89E-05	1,36E-05	7,80E-06
18	4,27E-04	1,32E-05	8,28E-05	7,42E-05	1,02E-04	5,23E-05	7,31E-05	1,53E-05	1,83E-04	1,11E-05
19	5,79E-05	5,52E-05	6,50E-08	0,00E+00	4,42E-08	1,13E-08	2,20E-05	2,11E-05	1,35E-05	2,11E-05
20	1,24E-05	8,19E-06	3,82E-08	4,54E-05	6,37E-08	5,27E-05	2,49E-05	2,41E-05	1,47E-05	2,17E-05
21	1,21E-05	1,91E-04	9,85E-09	2,18E-06	1,14E-08	1,11E-07	2,41E-05	1,74E-05	1,09E-05	1,74E-05
22	1,13E-05	5,49E-06	9,78E-09	4,54E-05	1,13E-08	0,00E+00	2,11E-05	2,49E-05	7,67E-06	1,36E-05
23	1,06E-05	9,49E-05	0,00E+00	2,28E-05	0,00E+00	1,26E-04	2,03E-05	2,03E-05	8,20E-06	3,29E-05
24	8,19E-06	3,51E-05	0,00E+00	4,55E-08	0,00E+00	0,00E+00	1,53E-05	1,99E-05	7,00E-06	1,03E-05
25	5,49E-06	3,83E-06	0,00E+00	0,00E+00	0,00E+00	0,00E+00	1,33E-05	1,22E-05	6,40E-06	7,00E-06
26	8,30E-06	2,94E-04	4,07E-12	2,42E-04	7,41E-11	1,46E-04	1,36E-05	7,31E-05	2,17E-05	1,56E-04
27	5,23E-06	5,23E-06	0,00E+00	0,00E+00	0,00E+00	0,00E+00	1,25E-05	1,25E-05	1,01E-05	1,01E-05
28	5,35E-06	5,35E-06	0,00E+00	0,00E+00	0,00E+00	0,00E+00	1,24E-05	1,24E-05	9,55E-06	9,55E-06
29	5,27E-06	5,27E-06	0,00E+00	0,00E+00	0,00E+00	0,00E+00	1,24E-05	1,24E-05	7,03E-06	7,03E-06
30	5,27E-06	5,27E-06	0,00E+00	0,00E+00	0,00E+00	0,00E+00	1,23E-05	1,23E+05	4,84E-06	4,84E-06
31	5,22E-06	5,22E-06	0,00E+00	0,00E+00	0,00E+00	0,00E+00	1,23E-05	1,23E+05	4,86E-06	4,86E-06
32	5,22E-06	5,22E-06	0,00E+00	0,00E+00	0,00E+00	0,00E+00	1,23E-05	1,23E+05	4,70E-06	4,70E-06

Tabla 4.21: VLIW-Potencia por registro y aplicación. Configuración C2 (parte 1).

	epic				**unepic**				
	N-Opt	Opt	Opt	N-Opt	N-Opt	Opt	Opt	N-Opt	
1	9,97E-05	9,97E-05	1,87E-04	1,84E-05	5,39E-06	5,39E-06	0,0002	1,64E-05	17
2	5,15E-05	5,15E-05	1,51E-05	1,87E-04	3,95E-06	3,95E-06	2,90E-6	0,0002	18
3	5,16E-05	1,84E-05	1,75E-05	5,00E-05	1,70E-04	0,0001	1,27E-05	2,66E-05	19
4	1,01E-4	1,58E-05	3,41E-05	3,48E-05	3,99E-05	5,15E-06	2,93E-05	3,57E-05	20
5	8,45E-05	6,77E-05	3,16E-05	3,97E-05	0,0001	3,26E-05	3,57E-05	2,90E-05	21
6	6,77E-05	1,65E-05	2,08E-05	3,41E-05	7,52E-05	3,99E-05	5,41E-06	1,27E-05	22
7	6,25E-05	2,28E-05	5,16E-05	3,16E-05	5,87E-05	2,12E-05	0,0001	1,07E-05	23
8	1,63E-05	3,97E-05	2,37E-05	2,25E-05	2,90E-06	1,88E-05	1,07E-05	5,15E-06	24
9	2,37E-05	2,25E-05	1,63E-05	2,08E-05	3,26E-05	5,87E-05	4,03E-06	4,03E-06	25
10	1,58E-05	3,48E-05	8,45E-05	2,28E-05	2,84E-05	1,64E-05	0,0001	5,41E-06	26
11	1,51E-05	1,62E-05	1,96E-05	1,96E-05	2,90E-06	2,84E-05	3,48E-06	3,48E-06	27
12	1,75E-05	6,25E-05	1,52E-05	1,52E-05	2,93E-05	2,05E-05	3,40E-06	3,40E-06	28
13	1,54E-05	1,57E-05	1,52E-05	1,52E-05	1,88E-05	2,90E-05	2,98E-06	2,98E-06	29
14	1,65E-05	5,00E-05	1,80E-05	1,80E-05	2,05E-05	2,66E-05	2,92E-06	2,92E-06	30
15	1,57E-05	1,54E-05	1,65E-05	1,65E-05	0,0001	2,90E-06	2,92E-06	2,92E-06	31
16	1,62E-05	1,01E-4	1,59E-05	1,59E-05	2,12E-05	1,70E-04	3,46E-06	3,46E-06	32
	cjpeg				**djpeg**				
	N-Opt	Opt	Opt	N-Opt	N-Opt	Opt	Opt	N-Opt	
1	5,58E-05	5,58E-05	2,09E-04	5,51E-05	1,45E-05	1,45E-05	1,53E-04	7,17E-05	17
2	5,21E-05	5,21E-05	5,09E-05	2,09E-04	1,43E-05	1,43E-05	1,41E-05	1,28E-04	18
3	1,04E-04	1,04E-04	5,34E-05	5,84E-05	1,53E-04	1,21E-04	5,11E-05	3,68E-05	19
4	9,78E-05	5,26E-05	5,41E-05	5,63E-05	4,64E-05	1,41E-05	3,20E-05	2,81E-05	20
5	9,22E-05	5,24E-05	5,30E-05	5,41E-05	4,67E-05	4,93E-05	4,64E-05	3,51E-05	21
6	9,11E-05	9,11E-05	8,20E-05	5,33E-05	5,32E-05	2,81E-05	4,53E-05	3,20E-05	22
7	1,26E-04	5,18E-05	5,21E-05	5,40E-05	1,21E-04	4,01E-05	3,43E-05	4,01E-05	23
8	5,09E-05	5,32E-05	5,33E-05	5,34E-05	1,41E-05	3,51E-05	5,32E-05	3,43E-05	24
9	8,20E-05	5,63E-05	5,10E-05	5,24E-05	5,31E-05	5,03E-05	2,03E-05	2,03E-05	25
10	5,21E-05	5,40E-05	9,22E-05	5,32E-05	5,11E-05	1,75E-05	7,17E-05	1,80E-05	26
11	5,10E-05	5,84E-05	5,27E-05	5,27E-05	1,41E-05	5,31E-05	2,33E-05	2,33E-05	27
12	5,30E-05	5,51E-05	5,18E-05	5,18E-05	4,53E-05	3,56E-05	2,36E-05	2,36E-05	28
13	5,26E-05	5,22E-05	5,17E-05	5,17E-05	3,56E-05	3,68E-05	1,86E-05	1,86E-05	29
14	5,22E-05	9,78E-05	5,19E-05	5,19E-05	5,03E-05	4,67E-05	1,77E-05	1,77E-05	30
15	5,18E-05	5,09E-05	5,16E-05	5,16E-05	4,93E-05	1,80E-05	1,77E-05	1,77E-05	31
16	5,09E-05	1,26E-04	5,15E-05	5,15E-05	1,75E-05	1,28E-04	1,75E-05	1,75E-05	32
	gsmdecode				**gsmencode**				
	N-Opt	Opt	Opt	N-Opt	N-Opt	Opt	Opt	N-Opt	
1	2,33E-06	2,33E-06	2,50E-04	2,64E-06	4,18E-06	4,18E-06	1,44E-04	3,51E-05	17
2	2,28E-06	2,28E-06	2,06E-06	1,06E-04	4,12E-06	4,12E-06	7,49E-06	4,27E-04	18
3	7,59E-05	1,06E-04	1,29E-05	1,38E-05	1,29E-04	1,91E-04	1,29E-04	5,79E-05	19
4	2,50E-04	2,67E-06	3,32E-05	1,17E-05	1,91E-04	8,30E-06	8,19E-06	1,24E-05	20
5	1,59E-04	7,59E-05	3,76E-05	7,79E-06	2,94E-04	5,52E-05	1,03E-04	1,21E-05	21
6	8,11E-05	2,05E-06	4,34E-06	6,03E-06	1,44E-04	3,51E-05	1,24E-05	1,13E-05	22
7	8,28E-05	8,28E-05	9,15E-05	4,34E-06	1,03E-04	9,09E-05	5,79E-05	1,06E-05	23
8	2,06E-06	7,79E-06	6,03E-06	2,67E-06	3,83E-06	1,32E-05	1,06E-05	8,19E-06	24
9	3,67E-05	1,38E-05	2,64E-06	2,55E-06	9,98E-05	1,08E-04	3,83E-06	5,49E-06	25
10	3,76E-05	6,08E-05	1,59E-04	2,37E-06	9,49E-05	5,49E-06	4,27E-04	8,30E-06	26
11	2,05E-06	2,55E-06	2,24E-06	2,24E-06	3,83E-06	9,98E-05	5,23E-06	5,23E-06	27
12	1,67E-04	8,11E-05	2,25E-06	2,25E-06	9,09E-05	1,13E-05	5,35E-06	5,35E-06	28
13	3,32E-05	1,17E-05	2,24E-06	2,24E-06	1,08E-04	9,49E-05	5,27E-06	5,27E-06	29
14	9,15E-05	3,67E-05	2,25E-06	2,25E-06	5,52E-05	1,21E-05	5,27E-06	5,27E-06	30
15	6,08E-05	2,37E-06	2,24E-06	2,24E-06	1,32E-05	3,83E-06	5,22E-06	5,22E-06	31
16	1,29E-05	1,67E-04	2,25E-06	2,25E-06	7,49E-06	2,94E-04	5,22E-06	5,22E-06	32

Tabla 4.22: VLIW-Potencia por registro y aplicación. Configuración C2 (parte 2).

	rawcaudio				rawdaudio				
	N-Opt	Opt	Opt	N-Opt	N-Opt	Opt	Opt	N-Opt	
1	0,00E+00	0,00E+00	2,53E-04	9,61E-08	0,00E+00	0,00E+00	2,43E-04	1,11E-07	17
2	7,41E-11	7,41E-11	0,00E+00	8,28E-05	7,41E-11	7,41E-11	0,00E+00	1,02E-04	18
3	6,91E-05	7,71E-05	0,00E+00	6,50E-08	7,40E-05	1,02E-04	2,22E-10	4,42E-08	19
4	2,42E-04	6,50E-08	9,78E-09	3,82E-08	2,43E-04	0,00E+00	0,00E+00	6,37E-08	20
5	2,53E-04	4,54E-05	4,54E-05	9,85E-09	1,46E-04	4,14E-05	5,23E-05	1,14E-08	21
6	7,71E-05	0,00E+00	4,55E-08	9,78E-09	1,26E-04	7,41E-11	1,13E-08	1,13E-08	22
7	7,42E-05	7,42E-05	2,28E-05	0,00E+00	5,27E-05	7,40E-05	5,23E-05	0,00E+00	23
8	0,00E+00	3,82E-08	0,00E+00	0,00E+00	0,00E+00	6,37E-08	4,42E-08	0,00E+00	24
9	4,54E-05	2,18E-06	7,41E-11	0,00E+00	5,23E-05	1,11E-07	0,00E+00	0,00E+00	25
10	4,54E-05	6,91E-05	2,42E-04	7,41E-11	5,23E-05	5,27E-05	1,26E-04	7,41E-11	26
11	0,00E+00	0,00E+00	0,00E+00	0,00E+00	0,00E+00	0,00E+00	0,00E+00	0,00E+00	27
12	4,54E-05	4,54E-05	0,00E+00	0,00E+00	5,23E-05	5,23E-05	0,00E+00	0,00E+00	28
13	4,54E-05	9,61E-08	0,00E+00	0,00E+00	4,14E-05	1,14E-08	0,00E+00	0,00E+00	29
14	2,28E-05	4,54E-05	0,00E+00	0,00E+00	2,62E-05	2,62E-05	0,00E+00	0,00E+00	30
15	2,18E-06	9,85E-09	0,00E+00	0,00E+00	2,22E-10	0,00E+00	0,00E+00	0,00E+00	31
16	4,55E-08	8,28E-05	0,00E+00	0,00E+00	0,00E+00	1,46E-04	0,00E+00	0,00E+00	32
	mpegdecode				**mpegencode**				
	N-Opt	Opt	Opt	N-Opt	N-Opt	Opt	Opt	N-Opt	
1	3,24E-05	3,24E-05	3,59E-05	3,59E-05	1,08E-05	1,08E-05	1,83E-04	1,36E-05	17
2	2,05E-05	2,05E-05	2,67E-05	7,31E-05	7,23E-06	7,23E-06	4,70E-06	1,83E-04	18
3	2,89E-05	2,41E-05	1,80E-05	2,20E-05	1,56E-04	1,56E-04	7,80E-06	1,35E-05	19
4	2,67E-05	2,38E-05	2,72E-05	2,49E-05	1,55E-04	4,64E-06	1,09E-05	1,47E-05	20
5	1,80E-05	1,45E-05	1,22E-05	2,41E-05	1,23E-04	1,12E-05	2,17E-05	1,09E-05	21
6	2,88E-05	2,49E-05	2,89E-05	2,11E-05	3,29E-05	1,43E-05	1,36E-05	7,67E-06	22
7	1,99E-05	1,53E-05	1,33E-05	2,03E-05	1,89E-05	1,35E-05	1,47E-05	8,20E-06	23
8	1,22E-05	2,20E-05	2,88E-05	1,53E-05	4,64E-06	1,03E-05	9,35E-06	7,00E-06	24
9	1,99E-05	1,74E-05	1,36E-05	1,33E-05	2,85E-05	3,29E-05	7,67E-06	6,40E-06	25
10	3,27E-05	2,11E-05	3,27E-05	1,36E-05	1,43E-05	8,20E-06	1,23E-04	2,17E-05	26
11	1,22E-05	1,99E-05	1,25E-05	1,25E-05	4,70E-06	2,85E-05	1,01E-05	1,01E-05	27
12	2,72E-05	1,99E-05	1,24E-05	1,24E-05	1,11E-05	1,11E-05	9,55E-06	9,55E-06	28
13	2,38E-05	1,77E-05	1,24E-05	1,24E-05	1,12E-05	1,89E-05	7,03E-06	7,03E-06	29
14	1,74E-05	2,03E-05	1,23E-05	1,23E-05	7,80E-06	7,00E-06	4,84E-06	4,84E-06	30
15	1,45E-05	1,22E-05	1,23E-05	1,23E-05	1,03E-05	6,40E-06	4,86E-06	4,86E-06	31
16	1,77E-05	7,31E-05	1,23E-05	1,23E-05	9,35E-06	1,55E-04	4,70E-06	4,70E-06	32

Tabla 4.23: VLIW-Potencia por registro y aplicación. Configuración C3 (parte 1).

	epic																	
	N-Opt	Opt	N-Opt	Opt	N-Opt	Opt	N-Opt	Opt	N-Opt	Opt	N-Opt	Opt	N-Opt	Opt	N-Opt	Opt		
1	9,97E-05	9,97E-05	8,45E-05	1,01E-04	2,37E-05	5,16E-05	1,54E-05	3,16E-05	1,84E-05	3,48E-05	3,97E-05	3,97E-05	2,08E-05	1,87E-04	1,52E-05	1,52E-05	29	
2	5,15E-05	5,15E-05	6,77E-05	1,54E-05	1,58E-05	1,65E-05	1,65E-05	1,75E-05	1,87E-04	2,28E-05	3,41E-05	2,25E-05	2,28E-05	1,51E-05	1,80E-05	1,80E-05	30	
3	1,55E-06	1,57E-05	6,25E-05	1,84E-05	1,51E-05	1,63E-05	1,57E-05	1,62E-05	6,67E-10	1,58E-05	3,16E-05	2,08E-05	1,96E-05	1,96E-05	1,65E-05	1,65E-05	31	
4	1,01E-04	2,37E-05	1,63E-05	3,41E-05	1,75E-05	6,77E-05	1,62E-05	8,45E-05	3,48E-05	6,25E-05	2,25E-05	5,00E-05	1,52E-05	1,52E-05	1,59E-05	1,59E-05	32	
	N-Opt	Opt	N-Opt	Opt	N-Opt	Opt	N-Opt	Opt	N-Opt	Opt	N-Opt	Opt	N-Opt	Opt	N-Opt	Opt		
	N-opt	Opt	N-opt	Opt	N-opt	Opt	N-opt	Opt	N-opt	Opt	N-opt	Opt	N-opt	Opt	N-opt	Opt		
1	5,39E-06	5,39E-06	1,26E-04	2,90E-05	3,26E-05	2,84E-05	1,88E-05	3,99E-05	1,64E-05	3,57E-05	2,90E-05	1,07E-04	4,03E-06	1,96E-04	2,98E-06	2,98E-06	29	
2	3,95E-06	3,95E-06	7,52E-05	1,27E-05	2,84E-05	2,12E-05	2,05E-05	1,64E-05	1,96E-04	1,88E-05	1,27E-05	4,03E-06	5,41E-06	2,90E-06	2,92E-06	2,92E-06	30	
3	1,70E-04	2,90E-06	5,87E-05	5,15E-06	2,90E-06	1,07E-05	1,07E-04	2,66E-05	2,66E-05	5,41E-06	1,07E-05	2,05E-05	3,48E-06	3,48E-06	2,92E-06	2,92E-06	31	
4	3,99E-05	1,70E-04	2,90E-06	1,26E-04	2,93E-05	5,87E-05	2,12E-05	3,26E-05	3,57E-05	7,52E-05	5,15E-06	2,93E-05	3,40E-06	3,40E-06	3,46E-06	3,46E-06	32	
	cjpeg																	
	N-Opt	Opt	N-Opt	Opt	N-Opt	Opt	N-Opt	Opt	N-Opt	Opt	N-Opt	Opt	N-Opt	Opt	N-Opt	Opt		
1	5,58E-05	5,58E-05	9,22E-05	5,41E-05	8,20E-05	5,40E-05	5,26E-05	9,11E-05	5,51E-05	5,84E-05	5,41E-05	5,63E-05	5,24E-05	1,26E-04	5,17E-05	5,17E-05	29	
2	5,21E-05	5,21E-05	9,11E-05	5,34E-05	5,21E-05	5,33E-05	5,22E-05	5,22E-05	2,09E-04	5,32E-05	5,33E-05	5,30E-05	5,32E-05	5,09E-05	5,19E-05	5,19E-05	30	
3	1,04E-04	5,09E-05	1,26E-04	5,10E-05	5,10E-05	5,21E-05	5,18E-05	5,26E-05	5,84E-05	5,24E-05	5,40E-05	5,18E-05	5,27E-05	5,27E-05	5,16E-05	5,16E-05	31	
4	9,78E-05	2,09E-04	5,09E-05	1,04E-04	5,30E-05	9,22E-05	5,09E-05	5,51E-05	5,63E-05	8,20E-05	5,34E-05	9,78E-05	5,18E-05	5,18E-05	5,15E-05	5,15E-05	32	
	djpeg																	
	N-Opt	Opt	N-Opt	Opt	N-Opt	Opt	N-Opt	Opt	N-Opt	Opt	N-Opt	Opt	N-Opt	Opt	N-Opt	Opt		
1	1,45E-05	1,45E-05	4,67E-05	7,17E-05	5,31E-05	4,01E-05	3,56E-05	4,67E-05	7,17E-05	5,11E-05	3,51E-05	5,31E-05	2,03E-05	5,32E-05	1,86E-05	1,86E-05	29	
2	1,43E-05	1,43E-05	5,32E-05	2,03E-05	5,11E-05	2,81E-05	5,03E-05	3,43E-05	1,28E-04	3,20E-05	3,20E-05	1,80E-05	1,80E-05	4,64E-05	1,77E-05	1,77E-05	30	
3	1,53E-04	1,41E-05	1,21E-04	3,56E-05	1,41E-05	1,41E-05	4,93E-05	3,68E-05	3,68E-05	3,51E-05	4,01E-05	1,75E-05	2,33E-05	2,33E-05	1,77E-05	1,77E-05	31	
4	4,64E-05	1,53E-04	1,41E-05	4,93E-05	4,53E-05	1,21E-04	1,75E-05	5,03E-05	2,81E-05	4,53E-05	3,43E-05	1,28E-04	2,36E-05	2,36E-05	1,75E-05	1,75E-05	32	
	gsmdec																	
	N-Opt	Opt	N-Opt	Opt	N-Opt	Opt	N-Opt	Opt	N-Opt	Opt	N-Opt	Opt	N-Opt	Opt	N-Opt	Opt		
1	2,33E-06	2,33E-06	1,59E-04	3,67E-05	3,67E-05	8,28E-05	3,32E-05	3,76E-05	2,64E-06	6,08E-05	7,79E-06	3,32E-05	2,55E-06	1,67E-04	2,24E-06	2,24E-06	29	
2	2,28E-06	2,28E-06	8,11E-05	1,38E-05	3,76E-05	4,34E-06	9,15E-05	1,29E-05	1,06E-04	1,17E-05	6,03E-06	7,79E-06	2,37E-06	2,05E-06	2,25E-06	2,25E-06	30	
3	7,59E-05	2,06E-06	8,28E-05	2,55E-06	2,05E-06	2,67E-06	6,08E-05	2,37E-06	1,38E-05	6,03E-06	4,34E-06	2,64E-06	2,24E-06	2,24E-06	2,24E-06	2,24E-06	31	
4	2,50E-04	2,50E-04	2,06E-06	9,15E-05	1,67E-04	7,59E-05	1,29E-05	1,06E-04	1,17E-05	8,11E-05	2,67E-06	1,59E-04	2,25E-06	2,25E-06	2,25E-06	2,25E-06	32	

Tabla 4.24: VLIW-Potencia por registro y aplicación. Configuración C3 (parte 2).

gsmenc																	
	N-Opt	Opt	N-Opt	Opt	N-Opt	Opt	N-Opt	Opt	N-Opt	Opt	N-Opt	Opt	N-Opt	Opt	N-Opt	Opt	
1	4,18E-06	4,18E-06	2,94E-04	1,91E-04	9,98E-05	9,09E-05	1,08E-04	1,29E-04	3,51E-05	5,52E-05	1,21E-05	9,49E-05	5,49E-06	2,94E-04	5,27E-06	5,27E-06	29
2	4,12E-06	4,12E-06	1,44E-04	5,49E-06	9,49E-05	1,24E-05	5,52E-05	7,49E-06	4,27E-04	3,51E-05	1,13E-05	8,19E-06	8,30E-06	3,83E-06	5,27E-06	5,27E-06	30
3	1,29E-04	3,83E-06	1,03E-04	1,13E-05	3,83E-06	1,32E-05	1,32E-05	1,06E-05	5,79E-05	1,21E-05	1,06E-05	8,30E-06	5,23E-06	5,23E-06	5,22E-06	5,22E-06	31
4	1,91E-04	4,27E-04	3,83E-06	5,79E-05	9,09E-05	1,08E-04	7,49E-06	9,98E-05	1,24E-05	1,03E-04	8,19E-06	1,44E-04	5,35E-06	5,35E-06	5,22E-06	5,22E-06	32
rawcaudio																	
	N-Opt	Opt	N-Opt	Opt	N-Opt	Opt	N-Opt	Opt	N-Opt	Opt	N-Opt	Opt	N-Opt	Opt	N-Opt	Opt	
1	0,00E+00	0,00E+00	2,53E-04	7,71E-05	4,54E-05	4,54E-05	4,54E-05	8,28E-05	9,61E-08	4,54E-05	9,85E-09	4,54E-05	0,00E+00	2,53E-04	0,00E+00	0,00E+00	29
2	7,41E-11	7,41E-11	7,71E-05	9,78E-09	4,54E-05	0,00E+00	2,28E-05	0,00E+00	8,28E-05	3,82E-08	9,78E-09	6,50E-08	7,41E-11	7,41E-11	0,00E+00	0,00E+00	30
3	6,91E-05	0,00E+00	7,42E-05	4,55E-08	0,00E+00	0,00E+00	2,18E-06	9,61E-08	6,50E-08	0,00E+00	0,00E+00	9,85E-09	0,00E+00	0,00E+00	0,00E+00	0,00E+00	31
4	2,42E-04	2,42E-04	0,00E+00	2,28E-05	4,54E-05	6,91E-05	4,55E-08	2,18E-06	3,82E-08	7,42E-05	0,00E+00	4,54E-05	0,00E+00	0,00E+00	0,00E+00	0,00E+00	32
rawdaudio																	
	N-Opt	Opt	N-Opt	Opt	N-Opt	Opt	N-Opt	Opt	N-Opt	Opt	N-Opt	Opt	N-Opt	Opt	N-Opt	Opt	
1	0,00E+00	0,00E+00	1,46E-04	1,02E-04	5,23E-05	4,14E-05	4,14E-05	5,23E-05	1,11E-07	1,26E-04	1,14E-08	5,23E-05	0,00E+00	2,43E-04	0,00E+00	0,00E+00	29
2	7,41E-11	7,41E-11	1,26E-04	0,00E+00	5,23E-05	4,42E-08	2,62E-05	7,41E-11	1,02E-04	0,00E+00	1,13E-08	2,22E-10	7,41E-11	0,00E+00	0,00E+00	0,00E+00	30
3	7,40E-05	0,00E+00	5,27E-05	0,00E+00	0,00E+00	0,00E+00	2,22E-10	6,37E-08	4,42E-08	1,14E-08	0,00E+00	1,13E-08	0,00E+00	0,00E+00	0,00E+00	0,00E+00	31
4	2,43E-04	1,46E-04	0,00E+00	2,62E-05	5,23E-05	7,40E-05	0,00E+00	5,23E-05	6,37E-08	1,11E-07	0,00E+00	5,27E-05	0,00E+00	0,00E+00	0,00E+00	0,00E+00	32
mpegdec																	
	N-Opt	Opt	N-Opt	Opt	N-Opt	Opt	N-Opt	Opt	N-Opt	Opt	N-Opt	Opt	N-Opt	Opt	N-Opt	Opt	
1	3,24E-05	3,24E-05	1,80E-05	2,67E-05	1,99E-05	2,89E-05	2,38E-05	2,72E-05	3,59E-05	2,88E-05	2,41E-05	2,20E-05	1,33E-05	7,31E-05	1,24E-05	1,24E-05	29
2	2,05E-05	2,05E-05	2,88E-05	1,74E-05	3,27E-05	1,33E-05	1,74E-05	1,53E-05	7,31E-05	1,45E-05	2,11E-05	1,99E-05	1,36E-05	1,22E-05	1,23E-05	1,23E-05	30
3	2,89E-05	1,22E-05	1,99E-05	1,80E-05	1,22E-05	1,99E-05	1,45E-05	1,36E-05	2,20E-05	2,03E-05	2,03E-05	1,77E-05	1,25E-05	1,25E-05	1,23E-05	1,23E-05	31
4	2,67E-05	3,59E-05	1,22E-05	2,49E-05	2,72E-05	2,41E-05	1,77E-05	3,27E-05	2,49E-05	2,11E-05	1,53E-05	2,38E-05	1,24E-05	1,24E-05	1,23E-05	1,23E-05	32
mpegenc																	
	N-Opt	Opt	N-Opt	Opt	N-Opt	Opt	N-Opt	Opt	N-Opt	Opt	N-Opt	Opt	N-Opt	Opt	N-Opt	Opt	
1	1,08E-05	1,08E-05	1,23E-04	1,23E-04	2,85E-05	1,89E-05	1,12E-05	1,55E-04	1,36E-05	1,43E-05	1,09E-05	1,56E-04	6,40E-06	2,85E-05	7,03E-06	7,03E-06	29
2	7,23E-06	7,23E-06	3,29E-05	7,00E-06	1,43E-05	1,35E-05	7,80E-06	6,40E-06	1,83E-04	1,09E-05	7,67E-06	4,70E-06	2,17E-05	1,36E-05	4,84E-06	4,84E-06	30
3	1,56E-04	4,64E-06	1,89E-05	7,67E-06	4,70E-06	1,47E-05	1,03E-05	8,20E-06	1,35E-05	1,03E-05	8,20E-06	7,80E-06	1,01E-05	1,01E-05	4,86E-06	4,86E-06	31
4	1,55E-04	1,83E-04	4,64E-06	1,12E-05	1,11E-05	2,17E-05	9,35E-06	9,35E-06	1,47E-05	3,29E-05	7,00E-06	1,11E-05	9,55E-06	9,55E-06	4,70E-06	4,70E-06	32

Tabla 4.25: ARM-Máxima potencia por registro y porcentaje diferencia de la versión optimizada. Configuración C1.

	epic		unepic		cjpeg		djpeg		gsmdec	
	N-Opt	Opt	N-Opt	Opt	N-Opt	Opt	N-Opt	Opt	N-Opt	Opt
1	1,93E-03	1,93E-03	2,66E-03	2,66E-03	5,35E-03	5,35E-03	5,09E-03	5,09E-03	1,79E-03	1,79E-03
2	6,60E-04	6,43E-07	1,33E-03	1,19E-06	8,78E-04	2,21E-05	5,48E-04	1,16E-05	6,69E-04	1,57E-05
3	4,53E-04	7,03E-05	6,81E-04	1,27E-04	3,82E-04	5,65E-05	2,54E-04	3,28E-05	5,54E-04	1,13E-04
4	3,04E-04	1,39E-04	6,06E-04	2,59E-04	2,68E-04	1,12E-04	1,73E-04	6,33E-05	8,81E-04	1,52E-04
5	1,14E-03	3,04E-04	3,86E-04	2,34E-04	6,01E-04	3,82E-04	1,50E-04	6,68E-05	2,31E-04	6,38E-05
6	1,33E-04	4,82E-05	2,34E-04	3,86E-04	3,87E-04	3,59E-04	1,52E-04	1,14E-04	1,13E-04	6,69E-04
7	1,28E-04	6,60E-04	1,89E-04	1,11E-04	3,59E-04	3,87E-04	2,00E-04	1,52E-04	7,39E-05	3,92E-05
8	6,92E-05	6,92E-05	1,27E-04	6,81E-04	1,72E-04	2,68E-04	1,14E-04	2,00E-04	6,38E-05	2,31E-04
9	1,39E-04	1,33E-04	2,59E-04	7,59E-05	1,12E-04	6,01E-04	7,51E-05	1,73E-04	3,92E-05	7,39E-05
10	7,03E-05	4,53E-04	7,59E-05	6,06E-04	2,22E-04	1,72E-04	6,33E-05	2,54E-04	1,57E-05	5,54E-04
11	4,82E-05	1,28E-04	1,11E-04	1,89E-04	7,00E-04	7,00E-04	6,68E-05	7,51E-05	1,70E-05	1,38E-04
12	6,43E-07	3,80E-06	1,19E-06	2,20E-05	5,65E-05	2,22E-04	1,16E-05	5,48E-04	1,38E-04	1,70E-05
13	3,80E-06	1,14E-03	2,20E-05	1,33E-03	2,21E-05	8,78E-04	3,28E-05	1,50E-04	1,52E-04	8,81E-04
14	1,24E-04	1,24E-04	2,04E-04	2,04E-04	3,14E-04	3,14E-04	3,70E-04	3,70E-04	1,30E-04	1,30E-04
15	2,21E-05	2,21E-05	3,38E-05	3,38E-05	4,90E-05	4,90E-05	1,55E-05	1,55E-05	9,44E-05	9,44E-05
16	4,09E-05	4,09E-05	5,66E-05	5,66E-05	5,18E-05	5,18E-05	2,67E-05	2,67E-05	1,03E-04	1,03E-04
	gsmencode		**rawcaudio**		**rawdaudio**		**mpegdecode**		**mpegencode**	
	N-Opt	Opt	N-Opt	Opt	N-Opt	Opt	N-Opt	Opt	N-Opt	Opt
1	1,77E-03	1,77E-03	2,26E-03	2,26E-03	2,99E-03	2,99E-03	2,22E-03	2,22E-03	2,49E-03	2,49E-03
2	6,65E-04	1,36E-05	5,15E-04	2,88E-07	2,47E-04	3,93E-07	1,63E-03	2,54E-05	3,47E-04	8,42E-06
3	5,56E-04	1,35E-04	3,53E-04	5,87E-07	6,93E-04	8,22E-07	2,82E-04	1,56E-04	1,07E-04	1,00E-04
4	8,78E-04	1,51E-04	4,87E-06	1,16E-04	6,58E-06	1,59E-04	1,76E-04	2,05E-04	1,00E-04	2,16E-04
5	2,29E-04	1,08E-04	1,77E-04	1,77E-04	2,86E-04	1,59E-04	3,13E-04	3,13E-04	5,50E-04	2,80E-04
6	1,08E-04	5,56E-04	1,66E-04	1,66E-04	1,61E-04	2,86E-04	2,05E-04	5,76E-05	2,16E-04	1,93E-04
7	7,21E-05	6,13E-05	3,92E-04	1,84E-04	1,59E-04	1,61E-04	1,56E-04	9,39E-04	1,93E-04	3,47E-04
8	6,13E-05	2,29E-04	1,84E-04	1,16E-04	1,59E-04	2,52E-04	5,76E-05	7,18E-05	2,80E-04	1,07E-04
9	3,77E-05	3,77E-05	1,16E-04	3,53E-04	2,52E-04	2,47E-04	9,39E-04	2,82E-04	4,52E-05	5,50E-04
10	1,36E-05	6,65E-04	1,16E-04	1,16E-04	2,52E-04	2,52E-04	2,54E-05	1,76E-04	1,22E-03	4,52E-05
11	1,68E-05	7,21E-05	1,16E-04	3,92E-04	3,02E-04	3,02E-04	2,78E-05	2,78E-05	5,12E-04	5,12E-04
12	1,35E-04	1,68E-05	2,88E-07	4,87E-06	3,93E-07	6,58E-06	7,18E-05	1,63E-03	1,61E-05	1,61E-05
13	1,51E-04	8,78E-04	5,87E-07	5,15E-04	8,22E-07	6,93E-04	3,45E-05	3,45E-05	8,42E-06	1,22E-03
14	1,26E-04	1,26E-04	1,94E-04	1,94E-04	2,66E-04	2,66E-04	3,10E-04	3,10E-04	9,70E-05	9,70E-05
15	9,41E-05	9,41E-05	4,38E-07	4,38E-07	5,89E-07	5,89E-07	7,31E-05	7,31E-05	1,51E-05	1,51E-05
16	1,01E-04	1,01E-04	1,33E-06	1,33E-06	1,82E-06	1,82E-06	7,13E-05	7,13E-05	1,48E-05	1,48E-05

A continuación, las tablas 4.25, 4.26, 4.27 muestran el comportamiento en cuanto al consumo de potencia para las configuraciones en ARM. Se comprueba como en la versión optimizada el MOEA ubica al lado de aquellos registros con mayor consumo de potencia otros con potencia menor para reducir el impacto térmico sobre el registro y el banco de registros. Al igual que para VLIW, se marcan en naranja algunas de las posiciones que rodean a valores de potencia comparativamente altos, en la distribución no optimizada. En la solución optimizada se marcan

en azul las posiciones que rodean el valor anteriormente marcado. Así, es posible identificar dónde ha situado el proceso de optimización los accesos a cada registro y los accesos de qué otros registros ha situado al lado, generalmente aquellos con menor consumo de potencia, favoreciendo la reducción de temperatura del banco de registros.

Para explicarlo, nos centramos en la tabla 4.25, por ejemplo, en la aplicación *mpegdecode*. Se observa como en la distribución no optimizada el valor $1,63E-03$ se encuentra entre los valores $2,22E-03$ y $2,82E-04$, ambos marcados en naranja. En la solución optimizada este valor se ha desplazado hasta la posición 12, alejándolo del valor $2,22E-03$ que en relación al resto de valores es alto y puede incrementar la temperatura del registro (debido a la transferencia de calor) y, por tanto, del banco de registros. Además, ha situado a su lado los valores $2,78E-05$ y $3,45E-05$, marcados en azul y que son, en comparación, significativamente más bajos al resto.

Finalmente, donde mejor se refleja la reducción de temperatura es en la gráfica que representa el mapa térmico del banco de registros para cada aplicación, tras el proceso de optimización. Como ejemplo de ello, la Figura 4.19 representa el mapa térmico de la configuración $C3$ de VLIW, solución aportada por el MOEA. En ella se puede observar, al comparar con la solución no optimizada de la Figura 4.13, que el incremento de temperatura es menor. De igual forma, la Figura 4.20 muestra el mapa térmico para la configuración $C3$ de ARM. Al comparar con la Figura 4.16, como se ha mencionado anteriormente, se observa que la distribución espacial realizada por el MOEA, separa los registros con mayor temperatura y permite reducir el impacto térmico sobre el banco de registros.

Tabla 4.26: ARM-Máxima potencia por registro y porcentaje diferencia de la versión optimizada.Configuración C2.

	epic				unepic				
	N-Opt	Opt	Opt	N-Opt	N-Opt	Opt	Opt	N-Opt	
1	1,93E-03	1,14E-03	1,93E-03	1,39E-04	2,66E-03	6,81E-04	2,66E-03	2,59E-04	9
2	6,60E-04	3,80E-06	6,43E-07	2,69E-07	1,33E-03	1,27E-04	1,19E-06	7,59E-05	10
3	4,53E-04	1,33E-04	4,82E-05	4,82E-05	6,81E-04	3,86E-04	7,59E-05	1,11E-04	11
4	3,04E-04	1,39E-04	7,03E-05	6,43E-07	6,06E-04	1,89E-04	1,11E-04	1,19E-06	12
5	1,14E-03	1,28E-04	4,53E-04	3,80E-06	3,86E-04	2,59E-04	6,06E-04	2,20E-05	13
6	1,33E-04	3,04E-04	1,24E-04	1,24E-04	2,34E-04	2,34E-04	2,04E-04	2,04E-04	14
7	1,28E-04	6,92E-05	2,21E-05	2,21E-05	1,89E-04	2,20E-05	3,38E-05	3,38E-05	15
8	6,92E-05	6,60E-04	4,09E-05	4,09E-05	1,27E-04	1,33E-03	5,66E-05	5,66E-05	16
	cjpeg				**djpeg**				
	N-Opt	Opt	Opt	N-Opt	N-Opt	Opt	Opt	N-Opt	
1	5,35E-03	7,00E-04	5,35E-03	1,12E-04	5,09E-03	2,54E-04	5,09E-03	7,51E-05	9
2	8,78E-04	3,59E-04	2,21E-05	2,22E-04	5,48E-04	2,00E-04	1,16E-05	6,33E-05	10
3	3,82E-04	3,82E-04	5,65E-05	7,00E-04	2,54E-04	1,14E-04	3,28E-05	6,68E-05	11
4	2,68E-04	3,87E-04	1,12E-04	5,65E-05	1,73E-04	1,73E-04	6,33E-05	1,16E-05	12
5	6,01E-04	2,22E-04	2,68E-04	2,21E-05	1,50E-04	1,50E-04	6,68E-05	3,28E-05	13
6	3,87E-04	6,01E-04	3,14E-04	3,14E-04	1,52E-04	1,52E-04	3,70E-04	3,70E-04	14
7	3,59E-04	1,72E-04	4,90E-05	4,90E-05	2,00E-04	7,51E-05	1,55E-05	1,55E-05	15
8	1,72E-04	8,78E-04	5,18E-05	5,18E-05	1,14E-04	5,48E-04	2,67E-05	2,67E-05	16
	gsmdecode				**gsmencode**				
	N-Opt	Opt	Opt	N-Opt	N-Opt	Opt	Opt	N-Opt	
1	1,79E-03	6,69E-04	1,79E-03	3,92E-05	1,77E-03	6,65E-04	1,77E-03	3,77E-05	9
2	6,69E-04	7,39E-05	1,57E-05	1,57E-05	6,65E-04	7,21E-05	1,36E-05	1,36E-05	10
3	5,54E-04	1,52E-04	6,38E-05	1,70E-05	5,56E-04	1,35E-04	6,13E-05	1,68E-05	11
4	8,81E-04	1,38E-04	1,70E-05	1,38E-04	8,78E-04	2,29E-04	1,68E-05	1,35E-04	12
5	2,31E-04	2,31E-04	5,54E-04	1,52E-04	2,29E-04	1,08E-04	5,56E-04	1,51E-04	13
6	1,13E-04	1,13E-04	1,30E-04	1,30E-04	1,08E-04	1,51E-04	1,26E-04	1,26E-04	14
7	7,39E-05	3,92E-05	9,44E-05	9,44E-05	7,21E-05	3,77E-05	9,41E-05	9,41E-05	15
8	6,38E-05	8,81E-04	1,03E-04	1,03E-04	6,13E-05	8,78E-04	1,01E-04	1,01E-04	16
	rawcaudio				**rawdaudio**				
	N-Opt	Opt	Opt	N-Opt	N-Opt	Opt	Opt	N-Opt	
1	2,26E-03	5,15E-04	2,26E-03	1,16E-04	2,99E-03	3,02E-04	2,99E-03	2,52E-04	9
2	5,15E-04	4,87E-06	2,88E-07	1,16E-04	2,47E-04	2,86E-04	3,93E-07	2,52E-04	10
3	3,53E-04	3,53E-04	5,87E-07	1,16E-04	6,93E-04	2,52E-04	8,22E-07	3,02E-04	11
4	4,87E-06	1,16E-04	1,16E-04	2,88E-07	6,58E-06	1,59E-04	6,58E-06	3,93E-07	12
5	1,77E-04	1,77E-04	1,66E-04	5,87E-07	2,86E-04	2,52E-04	2,47E-04	8,22E-07	13
6	1,66E-04	1,84E-04	1,94E-04	1,94E-04	1,61E-04	1,61E-04	2,66E-04	2,66E-04	14
7	3,92E-04	1,16E-04	4,38E-07	4,38E-07	1,59E-04	1,59E-04	5,89E-07	5,89E-07	15
8	1,84E-04	3,92E-04	1,33E-06	1,33E-06	1,59E-04	6,93E-04	1,82E-06	1,82E-06	16
	mpegdecode				**mpegencode**				
	N-Opt	Opt	Opt	N-Opt	N-Opt	Opt	Opt	N-Opt	
1	2,22E-03	1,63E-03	2,22E-03	9,39E-04	2,49E-03	5,50E-04	2,49E-03	2,80E-04	9
2	1,63E-03	2,78E-05	2,54E-05	2,54E-05	3,47E-04	1,07E-04	8,42E-06	4,52E-05	10
3	2,82E-04	7,18E-05	5,76E-05	2,78E-05	1,07E-04	3,47E-04	4,52E-05	1,22E-03	11
4	1,76E-04	2,05E-04	1,56E-04	7,18E-05	1,00E-04	1,93E-04	1,00E-04	5,12E-04	12
5	3,13E-04	1,76E-04	3,13E-04	3,45E-05	5,50E-04	2,80E-04	5,12E-04	1,61E-05	13
6	2,05E-04	2,82E-04	3,10E-04	3,10E-04	2,16E-04	2,16E-04	9,70E-05	8,42E-06	14
7	1,56E-04	3,45E-05	7,31E-05	7,31E-05	1,93E-04	1,61E-05	1,51E-05	9,70E-05	15
8	5,76E-05	9,39E-04	7,13E-05	7,13E-05	2,80E-04	1,22E-03	1,48E-05	1,51E-05	16

Tabla 4.27: ARM-Máxima potencia por registro y porcentaje diferencia de la versión optimizada. Configuración C3.

	1	3	5	7	9	11	13	15
	epic							
N-opt	1,93E-03	4,53E-04	1,14E-03	1,28E-04	1,39E-04	4,82E-05	3,80E-06	2,21E-05
Opt	1,14E-03	3,80E-06	4,82E-05	6,60E-04	4,53E-04	6,92E-05	1,28E-04	1,33E-04
Opt	3,04E-04	7,03E-05	6,43E-07	1,93E-03	1,39E-04	1,24E-04	2,21E-05	4,09E-05
N-opt	6,60E-04	3,04E-04	1,33E-04	6,92E-05	7,03E-05	6,43E-07	1,24E-04	4,09E-05
	unepic							
N-opt	2,66E-03	6,81E-04	3,86E-04	1,89E-04	2,59E-04	1,11E-04	2,20E-05	3,38E-05
Opt	1,19E-06	2,34E-04	6,06E-04	7,59E-05	3,86E-04	2,20E-05	2,59E-04	3,38E-05
Opt	2,66E-03	1,89E-04	1,11E-04	6,81E-04	1,27E-04	1,33E-03	2,04E-04	5,66E-05
N-opt	1,33E-03	6,06E-04	2,34E-04	1,27E-04	7,59E-05	1,19E-06	2,04E-04	5,66E-05
	cjpeg							
N-opt	5,35E-03	3,82E-04	6,01E-04	3,59E-04	1,12E-04	7,00E-04	2,21E-05	4,90E-05
Opt	2,21E-05	3,59E-04	3,82E-04	1,72E-04	7,00E-04	5,65E-05	3,87E-04	4,90E-05
Opt	5,35E-03	2,22E-04	2,68E-04	6,01E-04	1,12E-04	8,78E-04	3,14E-04	5,18E-05
N-opt	8,78E-04	2,68E-04	3,87E-04	1,72E-04	2,22E-04	5,65E-05	3,14E-04	5,18E-05
	djpeg							
N-opt	5,09E-03	2,54E-04	1,50E-04	2,00E-04	7,51E-05	6,68E-05	3,28E-05	1,55E-05
Opt	5,09E-03	7,51E-05	1,14E-04	6,68E-05	1,52E-04	5,48E-04	6,33E-05	1,55E-05
Opt	1,16E-05	1,50E-04	2,00E-04	2,54E-04	1,73E-04	3,28E-05	3,70E-04	2,67E-05
N-opt	5,48E-04	1,73E-04	1,52E-04	1,14E-04	6,33E-05	1,16E-05	3,70E-04	2,67E-05
	gsmdecode							
N-opt	1,79E-03	5,54E-04	2,31E-04	7,39E-05	3,92E-05	1,70E-05	1,52E-04	9,44E-05
Opt	1,57E-05	1,38E-04	5,54E-04	7,39E-05	6,69E-04	1,70E-05	1,52E-04	9,44E-05
Opt	1,79E-03	1,13E-04	6,38E-05	2,31E-04	3,92E-05	8,81E-04	1,30E-04	1,03E-04
N-opt	6,69E-04	8,81E-04	1,13E-04	6,38E-05	1,57E-05	1,38E-04	1,30E-04	1,03E-04
	gsmencode							
N-opt	1,77E-03	5,56E-04	2,29E-04	7,21E-05	3,77E-05	1,68E-05	1,51E-04	9,41E-05
Opt	1,77E-03	1,35E-04	1,68E-05	2,29E-04	6,13E-05	6,65E-04	1,51E-04	9,41E-05
Opt	1,36E-05	1,08E-04	8,78E-04	7,21E-05	5,56E-04	3,77E-05	1,26E-04	1,01E-04
N-opt	6,65E-04	8,78E-04	1,08E-04	6,13E-05	1,36E-05	1,35E-04	1,26E-04	1,01E-04
	rawcaudio							
N-opt	2,26E-03	3,53E-04	1,77E-04	3,92E-04	1,16E-04	1,16E-04	5,87E-07	4,38E-07
Opt	2,26E-03	1,16E-04	1,84E-04	4,87E-06	3,53E-04	5,87E-07	1,66E-04	4,38E-07
Opt	2,88E-07	1,77E-04	1,16E-04	3,92E-04	1,16E-04	5,15E-04	1,94E-04	1,33E-06
N-opt	5,15E-04	4,87E-06	1,66E-04	1,84E-04	1,16E-04	2,88E-07	1,94E-04	1,33E-06
	rawdaudio							
N-opt	2,99E-03	6,93E-04	2,86E-04	1,59E-04	2,52E-04	3,02E-04	8,22E-07	5,89E-07
Opt	3,93E-07	3,02E-04	2,52E-04	1,61E-04	6,93E-04	1,59E-04	2,47E-04	5,89E-07
Opt	2,99E-03	6,58E-06	1,59E-04	2,52E-04	8,22E-07	2,86E-04	2,66E-04	1,82E-06
N-opt	2,47E-04	6,58E-06	1,61E-04	1,59E-04	2,52E-04	3,93E-07	2,66E-04	1,82E-06
	mpegdecode							
N-opt	2,22E-03	2,82E-04	3,13E-04	1,56E-04	9,39E-04	2,78E-05	3,45E-05	7,31E-05
Opt	2,22E-03	5,76E-05	2,82E-04	3,45E-05	1,76E-04	1,63E-03	7,18E-05	7,31E-05
Opt	2,54E-05	3,13E-04	1,56E-04	9,39E-04	2,05E-04	2,78E-05	3,10E-04	7,13E-05
N-opt	1,63E-03	1,76E-04	2,05E-04	5,76E-05	2,54E-05	7,18E-05	3,10E-04	7,13E-05
	mpegencode							
N-opt	2,49E-03	1,07E-04	5,50E-04	1,93E-04	4,52E-05	5,12E-04	8,42E-06	1,51E-05
Opt	8,42E-06	2,80E-04	1,93E-04	3,47E-04	4,52E-05	1,22E-03	5,12E-04	1,51E-05
Opt	2,49E-03	1,07E-04	2,16E-04	1,00E-04	5,50E-04	1,61E-05	9,70E-05	1,48E-05
N-opt	3,47E-04	1,00E-04	2,16E-04	2,80E-04	1,22E-03	1,61E-05	9,70E-05	1,48E-05
	2	4	6	8	10	12	14	16

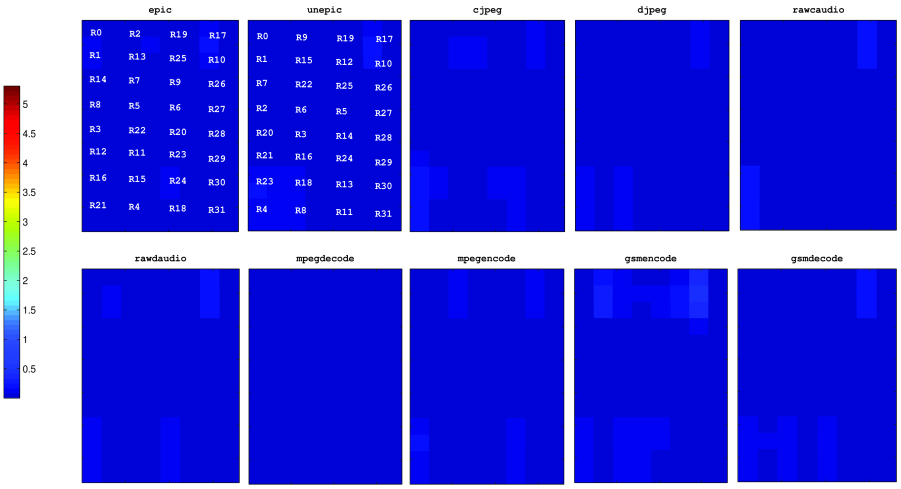


Figura 4.19: Mapa térmico para la configuración *C*3 sobre una arquitectura VLIW y tras el proceso de optimización mediante MOEA. Tras ubicar registros con un bajo impacto térmico al lado de otros con mayor impacto térmico, la temperatura de la estructura completa disminuye en todas las aplicaciones como se muestra también en la Tabla 4.7.

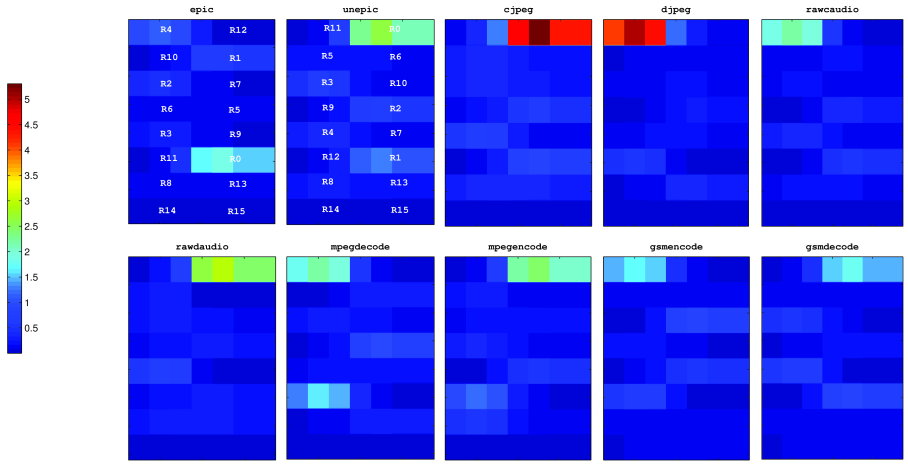


Figura 4.20: Mapa térmico para la configuración *C*3 sobre una arquitectura ARM y tras el proceso de optimización mediante MOEA. Tras ubicar registros con un bajo impacto térmico térmico al lado de otros con mayor impacto térmico, la temperatura de la estructura completa disminuye en todas las aplicaciones como se muestra también en la Tabla 4.8.

4.6. Conclusiones

En este capítulo se ha llevado a cabo el estudio térmico del banco de registros. Debido al alto número de accesos que soporta, se ha considerado digno de estudio el consumo energético debido a los accesos, así como la influencia sobre la tempe-

ratura del procesador. Estas características han llevado a introducir un conjunto de técnicas tanto hardware como software, con la finalidad de reducir la densidad de potencia y, por tanto, evitar valores altos de temperatura.

Con el objetivo de reducir el impacto térmico del banco de registros y, por consiguiente, el consumo de energía, se presenta un marco de optimización basado en computación evolutiva, perfilado estático de aplicaciones y el análisis de la temperatura en estado estable o estacionario, según el modelo térmico explicado en la sección 4.2.

Para abordar el estudio, se ha seleccionado el banco de registros de las arquitecturas VLIW y ARM (RISC), que presentan un comportamiento opuesto. El proceso de optimización está dirigido a aportar una distribución espacial de los accesos a los registros que componen la estructura del banco de registros, con el fin de situar lo más alejados posible aquellos registros cuya densidad de potencia y, por tanto, su temperatura sea mayor.

El proceso de optimización está dirigido por un MOEA, concretamente NSGA-II bi-objetivo, que recibe como entrada las trazas de las aplicaciones de forma independiente y tras evaluar la viabilidad física y la bondad de las soluciones obtenidas, obtiene una o un conjunto de soluciones para cada aplicación, donde cada una representa una distribución espacial del banco de registros.

Para realizar los experimentos se ha seleccionado un subconjunto de las aplicaciones Mediabench que recogen las principales características de las aplicaciones multimedia. Los resultados obtenidos muestran que el incremento de temperatura para ambas arquitecturas es bajo, es decir, el impacto térmico en el banco de registros no es significativo y por tanto, el proceso de optimización no aporta diferencias relevantes en valores absolutos, aunque en términos de % sí se detecta mejora. La principal razón de ello es que las optimizaciones realizadas por el compilador permiten reducir el impacto térmico. Sin embargo, se ha considerado que algunos de los valores obtenidos eran dignos de estudio y se ha procedido al proceso de optimización.

Los resultados de la optimización han demostrado que es posible reducir, aun más, el impacto térmico. Para todas las aplicaciones abordadas, las soluciones aportadas por el algoritmo multiobjetivo consiguen reducir la densidad de potencia y, en consecuencia, el incremento de temperatura del banco de registros en todas y cada una de las tres configuraciones estructurales y arquitecturas propuestas.

Capítulo 5

Optimización de la memoria cache

En el siguiente nivel de la jerarquía de memoria, tras los registros del procesador, se encuentra el subsistema de memoria cache que es el segundo objetivo a abordar en esta tesis. En este sentido, se propone un marco de optimización dirigido por tres técnicas evolutivas que tienen como objetivo la optimización del subsistema de memoria cache. En las siguientes secciones se realiza la introducción al problema y se presentan las partes que son comunes a los tres enfoques propuestos.

5.1. Marco teórico

El gran desarrollo experimentado por los sistemas empotrados en las últimas décadas, tal y como se ha mencionado en capítulos anteriores, los ha convertido en uno de los principales motores de la tecnología. Los dispositivos móviles como *smartphones*, cámaras digitales, navegadores GPS, etc. se actualizan de forma continua y añaden nuevas funcionalidades. La mayoría de las nuevas características incorporadas están dedicadas a dar soporte a nuevos servicios y aplicaciones multimedia, a pesar de la limitación en cuanto a recursos presentes en los sistemas empotrados. Una de estas limitaciones viene impuesta por el hecho de que los sistemas móviles tienen como fuente de alimentación una batería, cuya capacidad y tamaño vienen limitados por las restricciones de diseño.

Otro aspecto a destacar es la limitada capacidad de enfriamiento, lo cual define uno de los principales requisitos: un consumo reducido de energía. Además, los dispositivos móviles generalmente ejecutan aplicaciones multimedia, las cuales

requieren de alto rendimiento y, por tanto, consumen mucha energía. En consecuencia, uno de los principales objetivos de los diseñadores de sistemas empotrados es la búsqueda del equilibrio entre incrementar el rendimiento y reducir el consumo de energía, a bajo coste.

La optimización de la energía afecta directamente al envejecimiento de los transistores, que es uno de los factores que limitan la fiabilidad a largo plazo de los dispositivos. En un contexto común, la vida útil de un dispositivo se determina por el componente que falla más pronto, el impacto del envejecimiento es más serio en matrices de memoria, donde el fallo de una celda individual SRAM podría provocar el fallo de todo el dispositivo. Trabajos realizados previamente han mostrado que ahorrar energía en el subsistema de memoria puede controlar de forma efectiva los efectos del envejecimiento y puede prolongar significativamente la vida útil de la cache [170], [171]. Los enfoques que se presentan en esta tesis, optimizan rendimiento y energía y, por tanto, indirectamente mejoran la fiabilidad a largo plazo del dispositivo objetivo.

Por otra parte, hay que tener en cuenta que actualmente los sistemas empotrados se suministran con memorias cache que son lo suficientemente grandes como para influir sobre el rendimiento y el consumo de energía como nunca antes había sucedido en este tipo de sistemas. En el Capítulo 2 se mencionan algunos estudios que han identificado la memoria cache *on-chip* como uno de los componentes que más energía consume, situándose este consumo entre el 20 % y 30 % de la potencia total consumida por el chip en procesadores empotrados [5]. Adicionalmente, el diseño de la memoria cache tiene una influencia considerable sobre el consumo de energía y el rendimiento, debido a las diferencias en cuanto a complejidad hardware que ello implica. Por ejemplo, el diseño de una memoria cache directamente mapeada presenta el tiempo de acceso más rápido, pero necesita de mayor tamaño de cache para obtener un rendimiento comparable al de una cache asociativa por conjuntos de N-vías. Por otra parte, el diseño de memorias cache asociativas por conjuntos presenta un consumo de energía mayor, debido a que necesita acceder a un conjunto

de *N* etiquetas en cada operación [10].

Históricamente, el sistema de memoria cache se ha identificado como uno de los componentes que mejora tanto el rendimiento como el consumo de energía adaptando su configuración de acuerdo a los patrones de acceso a memoria de las aplicaciones que se están ejecutando. Sin embargo, encontrar una configuración óptima de cache para una única aplicación no es la mejor opción para otras aplicaciones que presentan patrones de acceso a memoria diferentes [10]. Adicionalmente, dado que las memorias cache disponen de muchos parámetros cuyo rango de posibles valores es muy amplio, los diseñadores se enfrentan a un espacio de búsqueda muy grande y cuya exploración consume mucho tiempo.

El comportamiento de la memoria cache no está únicamente condicionado por los parámetros estructurales (capacidad, tamaño de bloque, asociatividad, etc.), sino también por parámetros como los algoritmos de búsqueda, prebúsqueda y reemplazo, políticas de escritura, etc. Todas estas características conforman lo que se denomina una configuración de cache. La identificación de los valores óptimos para este conjunto de parámetros lleva a obtener el mejor rendimiento y consumo de energía y, en consecuencia, a identificar la mejor configuración de cache.

De esta forma, dado el comportamiento diferente de las aplicaciones, estas presentan requisitos de cache diferentes para alcanzar los objetivos específicos de rendimiento y de consumo de energía. Por tanto, se hace necesario tener en cuenta el comportamiento de las aplicaciones objetivo con el fin de encontrar la mejor configuración de cache que mejore el rendimiento y el consumo de energía.

En resumen, un esquema de optimización para determinar la mejor configuración de cache debería: (1) recorrer el espacio de diseño de las configuraciones de cache, que se define como el conjunto de todas las combinaciones de los valores, para todos los parámetros de cache; y (2) considerar la energía y el rendimiento de todas las aplicaciones objetivo para cada configuración de cache candidata, debido a los diferentes patrones de acceso a memoria.

Se podría pensar en abordar el problema con un primer enfoque de fuerza bruta

para obtener la mejor configuración de cache. Sin embargo, el proceso requiere de la ejecución y evaluación del tiempo y energía para todas las configuraciones de cache y aplicaciones objetivo, lo cual es una tarea difícil de llevar a cabo dado el tiempo reducido de lanzamiento al mercado. En este escenario, la obtención de tiempo y energía de una configuración de cache requiere su evaluación, para lo cual se necesita un simulador de cache capaz de procesar las trazas de los accesos a cache, obtenidas de una ejecución previa de las aplicaciones objetivo. Este proceso permitirá calcular el tiempo de ejecución y el consumo de energía para la aplicación en curso. Este es, generalmente, un proceso lento porque las trazas de programa guardan millones de operaciones de memoria que consumen decenas de segundos de ejecución del simulador. Por consiguiente, una búsqueda exhaustiva en este espacio de diseño sería una tarea inabordable, por la cantidad de tiempo necesaria para evaluar el alto número de configuraciones de cache que pueden ser definidas.

En la siguiente sección, se revisa el estado del arte para el problema de optimización cache.

5.2. Revisión de antecedentes

Está suficientemente demostrado que la memoria cache es un componente que juega un papel esencial para mejorar el rendimiento y el consumo de energía. En las últimas décadas, muchos investigadores han orientado su trabajo hacia la mejora de diferentes características de la memoria cache. Con el fin de incrementar el rendimiento y/o reducir el consumo de energía se han abordado diferentes enfoques de optimización, desarrollado herramientas de simulación y evaluación, etc. En esta tesis, se analizan alguno de estos trabajos que hemos separado en dos categorías: (1) trabajos que han abordado reconfiguración dinámica, y (2) trabajos basados en técnicas evolutivas. Ambas categorías han sido abordadas en el Capítulo 2 en la sección 2.2.2. A continuación se presentan principalmente las diferencias y/o inconvenientes respecto a la propuesta presentada en esta tesis.

5.2.1. Reconfiguración de cache

Como se ha mencionado previamente, uno de los principales objetivos durante muchos años ha sido configurar la memoria cache según el trabajo en ejecución, principalmente, para mejorar el rendimiento y/o el consumo de energía. Esto requiere tanto soporte hardware para la reconfiguración de cache como algoritmos eficientes para decidir los valores adecuados de los parámetros que consigan el mejor rendimiento, teniendo en cuenta la aplicación en ejecución. En este sentido, muchos trabajos de investigación han analizado los parámetros de memoria cache y han propuesto nuevas técnicas de reconfiguración desde diferentes puntos de vista.

En el estudio realizado por Naz et al. [59] donde la cache de datos se divide según el tipo de dato sea escalar o matricial, los parámetros vienen predefinidos antes de iniciar la optimización. Por tanto, el espacio de búsqueda en comparación con un espacio donde se tienen en cuenta todos los valores de los parámetros es reducido.

El enfoque propuesto por Givargis [172] busca mejorar el rendimiento de cache reduciendo el número de fallos en la cache de datos y de instrucciones, mientras que el objetivo en esta tesis es mejorar el rendimiento (tiempo de ejecución) y consumo de energía de forma simultánea.

En el enfoque presentado por Chen and Zou [60] se realiza un cambio automático de configuración según la fase en ejecución. El espacio de búsqueda está formado por tamaño de cache, tamaño de bloque y asociatividad. Desde el punto de vista de esta tesis, este trabajo no considera un número suficiente de parámetros.

Gordon-Ross et al. [61, 64, 62] proponen un algoritmo de gestión de reconfiguración eficiente para sistemas empotrados, ampliación al segundo nivel de cache desde un trabajo anterior y un método que primero aplica una fase de clasificación *off-line* para, posteriormente, aplicar un algoritmo de predicción on-line. En todos ellos, el espacio de búsqueda lo forman 3 parámetros: tamaño de cache, tamaño de bloque y asociatividad, y el número de valores para cada uno de los parámetros es reducido. En el caso de [64], la fase de predicción on-line requiere un número

pequeño de parámetros.

Wang et al. [67], basado en trabajos anteriores [65, 66], proponen reconfiguración dinámica de cache para sistemas empotrados en tiempo real. Minimizan el consumo de energía realizando un análisis estático en tiempo de ejecución. Sin embargo, el número de parámetros a optimizar era reducido: tamaño de cache (1, 2 o 4KB), tamaño de bloque (16, 32 o 64 bytes) y asociatividad (1-way, 2-way o 4-way). También se observa que estos enfoques minimizan el tiempo de ejecución o el consumo de energía. Adicionalmente, en este tesis se demuestra que una optimización basada en técnicas metaheurísticas fuera de línea puede encontrar valores óptimos de los parámetros de cache, sin necesidad de añadir complejidad hardware al diseño del subsistema de memoria estándar.

Más recientemente, la propuesta realizada en [58] minimiza el consumo de energía utilizando la técnica *way-concatenation* y permite reconfigurar la cache por software. Sin embargo, este enfoque suministra un número limitado de configuraciones de cache, permitiendo al ingeniero del sistema optimizar solamente tres parámetros: la asociatividad (1-way, 2-way o 4-way), el tamaño de cache y el tamaño de bloque.

Nuevas tecnologías de procesador basadas en núcleos como la propuesta realizada, por ejemplo, por ARM permiten cambiar la configuración para cada aplicación. El cambio de configuración afecta a los principales parámetros: tamaño de cache, tamaño de bloque y asociatividad. Sin embargo, para decidir qué configuración se adapta mejor a cada aplicación se considera necesario un algoritmo eficiente capaz de determinar qué valores son los óptimos para cada aplicación.

5.2.2. Perfilado estático

Con respecto al uso de perfilado estático, Andrade et al. [75] analizan el comportamiento de la cache en códigos con patrones irregulares, pero sólo abordan la optimización del tamaño de cache y la asociatividad, a diferencia del enfoque que aquí se propone. El enfoque propuesto por Rackesh Reddy [76], basado en el ma-

peo de tareas a diferentes particiones de cache para reducir el consumo de energía, es diferente al propuesto en esta tesis. Se trata de obtener el comportamiento de un conjunto de aplicaciones objetivo, obteniendo su perfil estático completo y la mejor configuración de la memoria cache (por ejemplo, tamaño, asociatividad, algoritmos de reemplazo y prebúsqueda, para la cache de instrucciones y de datos, y política de escritura para la cache de datos), para el conjunto completo.

Otros enfoques mencionados, como los presentados por Xingyan y Hongyan [77] o Feng et al. [78] se centran sólo en mejorar el algoritmo de reemplazo. El trabajo realizado por Gordon-Ross et al. [79] para estudiar la interacción del código de reordenamiento y la configuración de cache, como se ha mencionado, obtiene excelentes resultados. Sin embargo, esta técnica se aplicó únicamente a la cache de instrucciones y los métodos de optimización sistemáticos, que se presentan en esta tesis, se aplican a la configuración completa de las memorias cache de instrucciones y de datos.

5.2.3. Técnicas evolutivas

Las técnicas evolutivas han sido ampliamente aplicadas en la optimización tanto del diseño hardware como software y poniendo el foco en diferentes objetivos. La optimización multi-objetivo, por ejemplo, es ampliamente utilizada en problemas CAD (*Computer Aided Design*). En el Capítulo 2 en la sección 2.2.2, se han mencionado algunas propuestas en este sentido.

Los creadores de la técnica de Evolución Gramatical publicaron un trabajo donde se generaba automáticamente un algoritmo de reemplazo de cache [173]. Sin embargo, sólo consideraron la generación del algoritmo de reemplazo y mantenían fijos algunos parámetros como, por ejemplo, el tamaño de cache. De nuevo, se obviaron muchos de los parámetros anteriormente mencionados.

El enfoque de optimización multiobjetivo presentado por Palesi y Givargis [80] explora el espacio de diseño de una arquitectura SoC en busca de las configuraciones óptimas de Pareto del sistema. Sin embargo, el número de parámetros que

estudian es más pequeño que el presentado en esta tesis y su enfoque no es escalable a las actuales aplicaciones multimedia dado el bajo rendimiento de un SoC.

Risco et al. [89] aplican un algoritmo evolutivo multiobjetivo paralelo para optimizar aplicaciones de escritorio, para su uso en sistemas empotrados multimedia. El objetivo es diferente al presentado en esta tesis, la propuesta mejoraba el rendimiento, el uso de la memoria y el consumo de energía del subsistema de memoria.

Bui et al. [82] utilizan un GA sencillo para dar solución al problema de interferencia de cache, un enfoque diferente al que se propone en esta tesis. La optimización del rendimiento y el consumo de energía en un segundo nivel de cache, mediante un algoritmo basado en NSGA-II, fue propuesto por Filho et al. [81]. Sin embargo, de nuevo el número de parámetros es menor que en la propuesta que se realiza aquí, y se define por el tamaño de cache, tamaño de bloque y asociatividad.

Díaz et al. [174] aplican un sencillo GA en línea, pero sólo se aborda la optimización estudiando el parámetro de la asociatividad de cache y la mejora del rendimiento se dirigía a procesadores SMT y no a sistemas empotrados.

Dani et al. [83] proponen un algoritmo genético para encontrar una configuración óptima de cache para un sistema CMP. Sin embargo, no manejan un número mayor de parámetros de cache y como se ha mencionado anteriormente, el proceso de decodificación es complejo dado que codifican el chip y la cache en el cromosoma. Este problema hace difícil la aplicación del mismo método a una microarquitectura.

Como se ha visto, los enfoques que hacen uso de reconfiguración dinámica requieren complejidad hardware adicional en el diseño del subsistema de memoria. También, en la mayoría de los casos, esta complejidad añade una sobrecarga en tiempo de ejecución. En esta tesis se propone un enfoque diferente centrado en encontrar la mejor configuración de cache, después de explorar el espacio de búsqueda de diseños de cache. Esta estrategia evita la reconfiguración de cache y no añade complejidad hardware al diseño de memoria cache.

Adicionalmente, los trabajos anteriores definen su espacio de búsqueda median-

te el tamaño de cache, tamaño de bloque y asociatividad de la cache de instrucciones o de la cache de datos, pero no hemos encontrado ninguna propuesta que considere ambas cache al mismo tiempo en sistemas empotrados. La motivación de esta tesis es diferente desde que consideramos la optimización de la cache de instrucciones y de datos conjuntamente y seleccionamos todos los parámetros que pueden ser ajustados: tamaño de cache, tamaño de bloque, asociatividad, algoritmo de reemplazo y algoritmo de prebúsqueda para ambas cache; y política de escritura para la cache de datos. Por tanto, el espacio de búsqueda que se explora es más amplio.

Además, hasta donde conocemos, ninguno de los trabajos anteriores considera la optimización de tantos parámetros como se abordan en esta propuesta. Con respecto a los trabajos que usan técnicas evolutivas, la mayoría de los artículos citados se han centrado en explorar los típicos parámetros estructurales de la memoria cache como tamaño de cache, tamaño de bloque y asociatividad y además, normalmente tratan un rango pequeño de valores para estos parámetros. Una de las técnicas más utilizadas en esta clase de trabajos, algoritmos genéticos, presenta como inconveniente que el proceso de configuración debe ser personalizado para cada tipo de cache diferente y software de configuración.

5.3. Metodología

Tal y como se ha mencionado en las secciones anteriores, en el marco de optimización propuesto se aborda la mejora del rendimiento (tiempo de ejecución) y la reducción del consumo de energía, en relación con las operaciones sobre la memoria cache. Para abordar estos procesos de optimización es necesario, por tanto, seleccionar un modelo de rendimiento y energía que permita calcular tanto el tiempo de ejecución como el consumo de energía. A continuación se describe el modelo seleccionado que se aplica en cada uno de los enfoques que se proponen.

5.3.1. Modelo de rendimiento y energía

En los diferentes enfoques de optimización que se presentan en esta tesis, se considera el tiempo de ejecución y el consumo de energía de una configuración de cache dada cuando se está ejecutando una aplicación. Como se explicará más adelante, los procesos de optimización abordados necesitan de las trazas de programa que, posteriormente, deberán de ser procesadas por un simulador de cache que devuelva el comportamiento de cache, calculando el número de aciertos y fallos de cada ejecución.

Para llevar a cabo cada uno de los enfoques que aquí se proponen, se ha seleccionado la familia de procesadores ARM9 como sistema empotrado objetivo, donde será optimizada la configuración de cache. El diseño consiste en una memoria cache de primer nivel "L1" con cache de instrucciones y datos separadas y una memoria empotrada DRAM que actúa como memoria principal, cuyas características se muestran en la Tabla 5.1. Como se verá más adelante, esta metodología será extensible a varios niveles.

En este diseño, la cache de instrucciones es de sólo lectura. Las características de la memoria principal han de tenerse en cuenta para la evaluación de cada configuración de cache, en cada uno de los enfoques propuestos.

Tabla 5.1: Características de la memoria principal (DRAM).

Tamaño	64 Mb
Tiempo de acceso	$3{,}9889 \times 10^{-9}$ seg.
Ancho de banda	$6{,}7108864 \times 10^{9}$ bytes/seg.
Energía de acceso	1,051 W

Una vez que se han establecido las características hardware, se necesita un modelo capaz de traducir el número de aciertos y fallos de las aplicaciones ejecutadas, en tiempo de ejecución y consumo de energía. Para ello, se han seleccionado los modelos de rendimiento y energía descritos en [74], válidos para los entornos de optimización propuestos.

Los autores del artículo citado demuestran, con un 100 % de precisión, que los

modelos son equivalentes a los resultados obtenidos desde un simulador de cache. Por tanto, dado este rendimiento y teniendo en cuenta que las expresiones de los modelos pueden ser evaluadas de forma rápida, estos modelos encajan perfectamente en un algoritmo de optimización como los que se presentan en este capítulo.

Por tanto, considerando estos modelos, se estima el modelo de rendimiento y energía del subsistema de memoria cache. El modelo del resto de componentes del chip (CPU, memoria principal, etc.) no se tiene en cuenta dado que el esquema de optimización aborda sólo el subsistema de memoria cache. A continuación, se describen breve e individualmente los modelos de rendimiento y energía.

Modelo de rendimiento

El modelo de rendimiento para la memoria cache permite obtener el tiempo de ejecución. Este modelo se basa en el número de aciertos y fallos en el sistema de memoria cache y el tiempo necesario para resolverlos. La Ecuación (5.1) muestra como se calcula el tiempo de ejecución. Cada uno de sus componentes se describe debajo, aunque una explicación más amplia y detallada se puede encontrar en [74].

$$\begin{aligned} T = {} & Icache_{access} \times Icache_{access\_time} + Icache_{miss} \times DRAM_{access\_time} + \\ & Icache_{miss} \times Icache_{line\_size} \times \frac{1}{DRAM_{bwidth}} + \\ & Dcache_{access} \times Dcache_{access\_time} + Dcache_{miss} \times DRAM_{access\_time} + \\ & Dcache_{miss} \times Dcache_{line\_size} \times \frac{1}{DRAM_{bwidth}} \end{aligned} \tag{5.1}$$

Cada término en la ecuación representa:

- $Icache_{access}$ y $Dcache_{access}$ son el número de accesos a memoria cache de instrucciones y datos, respectivamente.
- $Icache_{miss}$ y $Dcache_{miss}$ se refieren al número de fallos de cache. Para cada uno, los datos deben ser copiados desde la memoria principal.
- $Icache_{access\_time}$ y $Dcache_{access\_time}$ representan el tiempo necesario para cada acceso a la cache de instrucciones y datos, respectivamente.

- $DRAM_{access\_time}$ es la latencia de la memoria principal.
- $Icache_{line\_size}$ y $Dcache_{line\_size}$ corresponden al tamaño de línea (tamaño de bloque) para la cache de instrucciones y de datos, respectivamente.
- *DRAM_bwidth* es la capacidad de transferencia de la DRAM.

De esta forma, la primera y cuarta parte en esta ecuación representada por $Icache_{access} \times Icache_{access\_time}$ y $Dcache_{access} \times Dcache_{access\_time}$ calculan el tiempo total necesario para resolver todos los accesos a la cache de instrucciones y datos, respectivamente. $Icache_{miss} \times DRAM_{access\_time}$ y $Dcache_{miss} \times DRAM_{access\_time}$ son el tiempo total consumido por los accesos a memoria principal, en respuesta a los fallos de la cache de instrucciones y de datos. La tercera y sexta parte, especificadas por $Icache_{miss} \times Icache_{line\_size} \times \frac{1}{DRAM\_bwidth}$ y $Dcache_{miss} \times Dcache_{line\_size} \times \frac{1}{DRAM\_bwidth}$ calculan el tiempo total necesario para rellenar una línea de la cache de instrucciones y datos cuando se produce un fallo, respectivamente.

Por tanto, la Ecuación (5.1), permite calcular el tiempo de ejecución de un programa a partir de su número de aciertos y fallos, teniendo en cuenta las características de cache.

5.3.2. Modelo de energía

El modelo de energía se define en la Ecuación (5.2) y también se detalla en [74]. Esta ecuación se utiliza para determinar el consumo de energía para una configura-

ción de cache.

$$
\begin{aligned}
E = {} & Total_{time} \times CPU_{power} + Icache_{access} \times Icache_{access\_energy} + \\
& Dcache_{access} \times Dcache_{access\_energy} + \\
& Icache_{miss} \times Icache_{access\_energy} \times Icache_{line\_size} + \\
& Dcache_{miss} \times Dcache_{access\_energy} \times Dcache_{line\_size} + \\
& Icache_{miss} \times DRAM_{access\_power} + \\
& (DRAM_{access\_time} + Icache_{line\_size} \times \frac{1}{DRAM_{bwidth}}) + \\
& Dcache_{miss} \times DRAM_{access\_power} + \\
& (DRAM_{access\_time} + Dcache_{line\_size} \times \frac{1}{DRAM_{bwidth}})
\end{aligned} \tag{5.2}
$$

La mayoría de los componentes de la ecuación ya se han descrito brevemente arriba. El resto de componentes se describen a continuación:

- $DRAM_{access\_power}$ representa la energía consumida en cada acceso a DRAM, acceso que se produce cada vez que sucede un fallo en la memoria cache.
- $Icache_{access\_energy}$ y $Dcache_{access\_energy}$ corresponden al consumo de energía para cada acceso a la cache de instrucciones y datos, respectivamente.

Los términos $Icache_{access} \times Icache_{access\_energy}$ y $Dcache_{access} \times Dcache_{access\_energy}$ calculan el consumo de energía debido a las memorias cache de instrucciones y datos, respectivamente. $Icache_{miss} \times Icache_{access\_energy} \times Icache_{line\_size}$ y $Dcache_{miss} \times Dcache_{access\_energy} \times Dcache_{line\_size}$ es el coste en términos de energía, de escribir datos a la cache de instrucciones y datos, cuando se produce un fallo. Los dos últimos términos calculan el coste de energía de la DRAM en respuesta a los fallos de cache. En este enfoque, se ha eliminado el primer término de la ecuación de la energía $Total_{time} \times CPU_{power}$ debido a tres razones: (1) el término CPU_{power} es constante y el término $Total_{time}$ corresponde al tiempo total del sistema, no sólo al subsistema de memoria cache, (2) $Total_{time} \times CPU_{power}$ representa la cantidad de energía consumida por la CPU y aquí se está optimizando sólo el rendimiento y la energía

consumidos por el subsistema de memoria, y (3) este término podría ser suficientemente grande para ocultar las diferencias debidas a las operaciones de memoria cache. Así, la ecuación de la energía se reduce a:

$$
\begin{aligned}
E = {} & Icache_{access} \times Icache_{access\_energy} + Dcache_{access} \times Dcache_{access\_energy} + \\
& Icache_{miss} \times Icache_{access\_energy} \times Icache_{line\_size} + \\
& Dcache_{miss} \times Dcache_{access\_energy} \times Dcache_{line\_size} + \\
& Icache_{miss} \times DRAM_{access\_power} + (DRAM_{access\_time} + \\
& Icache_{line\_size} \times \frac{1}{DRAM_{bwidth}}) + \\
& Dcache_{miss} \times DRAM_{access\_power} + (DRAM_{access\_time} + \\
& Dcache_{line\_size} \times \frac{1}{DRAM_{bwidth}})
\end{aligned}
\tag{5.3}
$$

Por tanto, la Ecuación (5.3), permite calcular el consumo de energía de un programa a partir de su número de aciertos y fallos, teniendo en cuenta las características de cache.

Todas las ecuaciones usan segundos, vatios, julios, bytes y bytes/seg como unidades para medir tiempo, potencia, energía, tamaño de línea de cache y ancho de banda, respectivamente.

La siguiente sección describe los principales problemas sobre el diseño de memorias cache y define el espacio de búsqueda del problema.

5.3.3. Diseño de cache y espacio de búsqueda

Con el fin de clarificar el alcance del problema en estudio, esta sección se dedica a introducir algunos de los conceptos del diseño de memorias cache y problemas asociados que deben ser abordados. Estos conceptos ayudarán a determinar el tamaño del espacio de búsqueda.

En primer lugar, se debe tener en cuenta que existen diferentes características de diseño, también denominados parámetros de diseño, que necesitan ser afinados con el fin de determinar una configuración de cache. Los valores que se asignen a esos parámetros influyen en el comportamiento de la memoria cache, tanto en tiempo de

ejecución como en el consumo de energía. Sin embargo, estos parámetros pueden ser diferentes dependiendo del procesador donde se incluirá la memoria cache.

En este sentido, se ha de decidir primero qué tipo de memoria cache seleccionar. Teniendo en cuenta que este enfoque se centra en sistemas empotrados, se ha seleccionado la familia de procesadores ARM9. Los procesadores ARM, debido a su bajo coste y consumo de energía, tienen una posición predominante en el mercado de los dispositivos móviles y se ha generalizado su presencia en dispositivos empotrados multimedia. De hecho, los procesadores de Apple A7, A8 y A8X incluidos en los populares iPhone 5S, iPhone 6 y iPad Air 2 implementan el conjunto de instrucciones de la versión ARMv8. Además, la familia de procesadores ARM9 está también presentes en dispositivos como por ejemplo, consolas de videojuegos (GP32, Nintendo DS), calculadora (HP 49/50), teléfonos móviles (Series HTC TyTN, K y W de Sony Ericsson) y Navegadores GPS (Samsung S3C2410 incorporado por el conocido dispositivo TomTom).

Los procesadores pertenecientes a esta familia de ARM permiten la implementación de una memoria cache con tamaños entre 4 KB y 128 KB. Por ejemplo, los procesadores ARM920T, ARM922T y ARM940T de la familia ARM9TDMI implementan memorias cache separadas de instrucciones y de datos de 16 KB, 8 KB y 4 KB, respectivamente. Sin embargo, las revisiones realizadas en la familia ARM9E han posibilitado el uso de tamaños superiores a 1 MB, como es el caso del procesador ARM946-S. Dado que la separación entre la cache de instrucciones y de datos es común en el diseño de los procesadores más recientes, en esta tesis se considera esta implementación de dos memorias cache separadas.

Por tanto, los parámetros de cache en el marco de optimización propuesto que pueden ser ajustados en las memorias cache de instrucciones y de datos son los siguientes: tamaño de cache, tamaño de bloque, algoritmo de reemplazo, asociatividad y algoritmo de prebúsqueda para ambas, cache de instrucciones y datos, y política de escritura para la cache de datos. De esta forma, el tamaño de cache es la capacidad de la memoria cache en bytes, el tamaño de bloque representa la canti-

dad de datos leídos y escritos en cada acceso a cache, el algoritmo de reemplazo se encarga de tomar la decisión en cuanto al bloque de datos a desalojar de la memoria cache y el algoritmo de prebúsqueda decide cómo se llevan los bloques a la memoria cache. Finalmente, la política de escritura decide cuándo los datos almacenados en la memoria cache deberían ser escritos en la memoria principal.

En resumen, el problema que se aborda consiste en seleccionar los valores para los parámetros de cache que optimizan el tiempo de ejecución y el consumo de energía de una aplicación dada. Así, el espacio de búsqueda está formado por todas las combinaciones de los diferentes valores para cada uno de los parámetros de cache. De esta forma, en el sistema objetivo, una configuración de cache está formada por 11 parámetros: 5 pertenecientes a la cache de instrucciones y 6 a la cache de datos. Según las configuraciones típicas de cache de los sistemas empotrados se selecciona un rango de valores permitidos para cada parámetro, los cuales se muestran en la Figura 5.1.

Para la cache de instrucciones (I-Cache), el tamaño de memoria puede tomar 8 valores diferentes comprendidos entre 512 bytes y 64 KB, el tamaño de bloque puede tomar 4 valores diferentes, es posible seleccionar entre 3 algoritmos de reemplazo y 3 algoritmos de prebúsqueda, y se dispone de 8 posibles valores para el nivel de asociatividad. El resultado son 2304 posibles configuraciones para la cache de instrucciones.

La cache de datos (D-Cache) tiene los mismos parámetros que la cache de instrucciones, más 2 políticas de escritura. Por tanto, se pueden seleccionar 4608 posibles configuraciones de cache para la cache de datos.

De esta manera, considerando todas las posibles combinaciones de ambas memorias cache, el tamaño del espacio de búsqueda es $2304 \times 4608 = 10616832$ configuraciones. Sin embargo, no todas ellas son posibles, dado que existen limitaciones que se refieren a combinaciones de parámetros no factibles. Por ejemplo, una configuración de cache de 512 Bytes, asociativa por conjunto de 64 vías con tamaño de bloque de 32 bytes no es factible. Con estos datos, el número de bloques se cal-

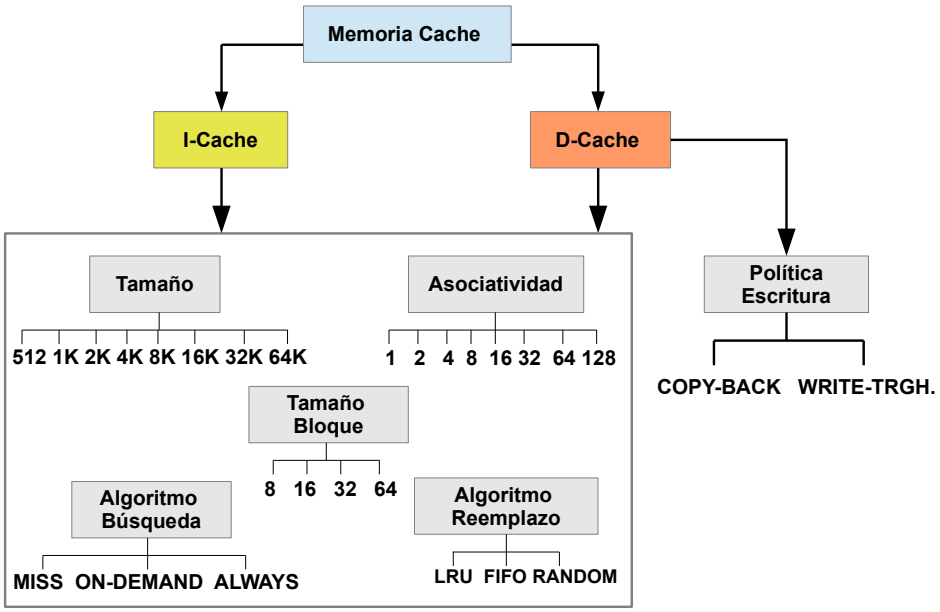


Figura 5.1: Parámetros de configuración de cache. Cache de instrucciones y de datos deben ser parametrizadas con los valores disponibles.

cula dividiendo el tamaño de la memoria cache entre el tamaño de bloque, en este caso el resultado es 16. De igual forma, el número de conjuntos es el resultado de dividir el número de bloques entre el número de vías que da un valor inferior a 1. Es decir, $Tam_{Cache}/(Tam_{Bloque} \times Asociatividad)$ da como resultado $0,25$, valor menor a 1 para un diseño de cache asociativa. Por tanto, los valores de los parámetros no son factibles. Entonces, si esta combinación se selecciona para la cache de instrucciones o de datos, debe ser considerada como combinación inválida por cualquier algoritmo de optimización y por tanto, no será evaluada.

El marco de optimización de cache implementado está dirigido por tres técnicas metaheurísticas diferentes dentro de una metodología común que se detalla en la siguiente sección.

5.3.4. Descripción de la metodología

Como se ha explicado anteriormente, en esta tesis se propone un marco de optimización con tres algoritmos evolutivos diferentes para identificar la mejor configuración de cache para un conjunto de aplicaciones objetivo. Se define la mejor configuración de cache como aquella que minimiza el tiempo de ejecución y el con-

sumo de energía para una aplicación dada o un conjunto de aplicaciones completo. Se presentan dos enfoques mono-objetivo dirigidos por GE para cada aplicación individual, y por DE para el conjunto de aplicaciones seleccionado. La propuesta restante es un enfoque multi-objetivo con NSGA-II que optimiza las aplicaciones de forma independiente.

Con esta finalidad, en esta sección se describe el entorno que se propone para optimizar la memoria cache, en este caso, el objetivo son los sistemas empotrados multimedia. La Figura 5.2 presenta un resumen del proceso de optimización completo y la metodología utilizada en cada objetivo parcial.

El proceso de optimización se divide en tres etapas diferentes. En primer lugar, dos procesos se ejecutan sólo una vez antes de la optimización. Estos se denominan procesos *off-line* y realizan las tareas de: (1) caracterización de la memoria cache y (2) generación de las trazas de las aplicaciones. La tercera etapa es la tarea de optimización, que está formada por el proceso evolutivo que en cada enfoque propuesto utiliza un algoritmo evolutivo diferente. El proceso evolutivo se encarga de llamar al simulador de cache, Dinero IV.

La optimización se realiza mediante el módulo correspondiente al EA de cada enfoque, como se ve en la Figura 5.2[1] . Cada vez que una configuración tiene que ser evaluada, la técnica evolutiva utilizada llama a Dinero IV [175], que es un simulador de cache conducido por trazas. Dinero IV recibe una configuración de cache desde el módulo del algoritmo de optimización y devuelve el número de aciertos y fallos para la traza de la aplicación objetivo. Estos datos se incluyen en los modelos de rendimiento y energía para evaluar la configuración. A continuación, se aportan más detalles de este flujo de optimización.

Procesos *off-line*

En este proceso se ejecutan dos tareas de forma *off-line*: la caracterización de todas las posibles configuraciones de cache y el perfilado del conjunto de aplicaciones objetivo. Ambas tareas son lentas, pero sólo deben ser ejecutadas una vez. Los

[1] * se explicará en la sección dedicada a DE

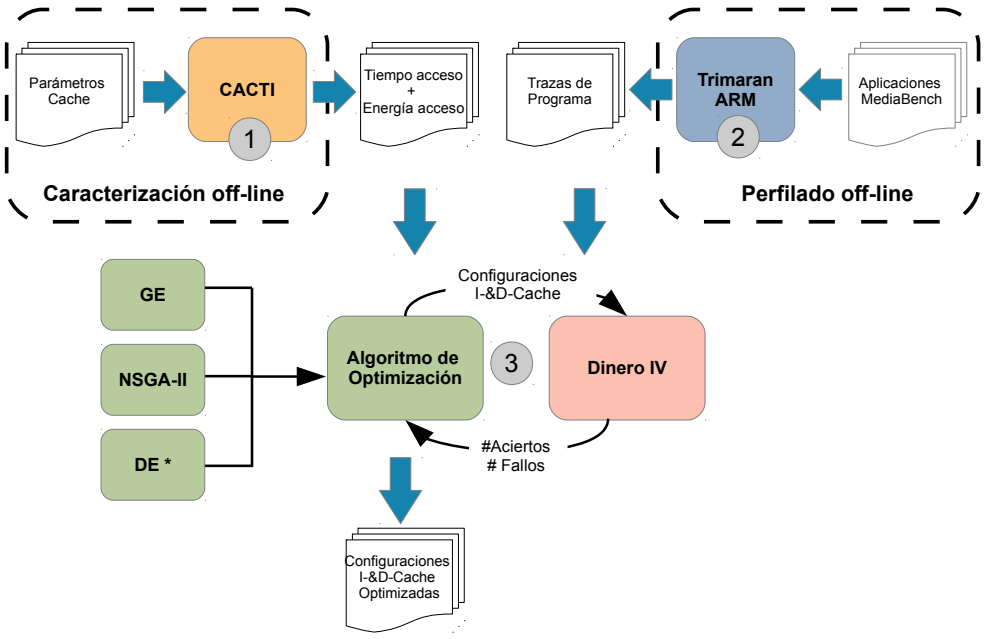


Figura 5.2: Procesos involucrados en la optimización de configuración de cache. El proceso se divide en tres etapas donde las dos primeras son previas al proceso de optimización, se realizan *off-line* y se encargan de la caracterización de la memoria cache y de generar las trazas de las aplicaciones. Sólo es necesario ejecutar una vez las dos primeras etapas para cada conjunto de parámetros estructurales de memoria cache y aplicaciones objetivo. La tercera etapa se encarga del proceso evolutivo donde cada marco de optimización utiliza una técnica metaheurística diferente que a su vez conlleva la ejecución del simulador de cache. Las técnicas metaheurísticas utilizadas son GE, NSGA-II y DE.

resultados se almacenan en ficheros de datos que el simulador de cache y el módulo correspondiente al algoritmo de optimización leen durante la optimización.

El proceso de caracterización, en la parte superior izquierda de la Figura 5.2, tiene la tarea específica de combinar todos los valores de los parámetros para las configuraciones de cache. Es decir, genera todas las posibles configuraciones de cache y obtiene su tiempo de acceso y el consumo de energía por acceso. Por lo tanto, este proceso suministra los valores de tiempo y energía usados en las ecuaciones (5.1) y (5.3), según los modelos de rendimiento y energía descritos en la Sección 5.3.1. Con este propósito, se utiliza CACTI donde se ha seleccionado una tecnología de 32*nm* para calcular el rendimiento y área de las memorias cache. Esta caracterización se ejecuta también sólo una vez. Para esta tarea, se ha seleccionado la herramienta web de la organización Cacti[176]. Los parámetros más importantes

requeridos por CACTI son aquellos relativos al hardware: tamaño de cache, tamaño de bloque y asociatividad; sin distinción entre cache de instrucciones y datos. A pesar de ser una tarea lenta es fácilmente abordable.

El proceso de perfilado, en la parte superior derecha de la Figura 5.2, se encarga de la generación de las trazas de las aplicaciones objetivo. De nuevo, este es un proceso que sólo se ejecuta una vez. La técnica de análisis toma cada aplicación y obtiene la secuencia de aciertos y fallos de cache durante la ejecución y genera un fichero de trazas con las operaciones de memoria. El proceso se lleva a cabo por la infraestructura de monitorización del rendimiento y compilador de código abierto denominado Trimaran. Trimaran proporciona los recursos necesarios para obtener trazas de aplicación con precisión. Dado que los procesadores ARM no están directamente soportados por Trimaran, se ha modificado para incorporar código desde el simulador arquitectónico SimpleScalar, que es capaz de modelar un amplio conjunto de arquitecturas diferentes, estando ARM entre ellas.

El fichero de traza es representativo de la ejecución de cada aplicación porque en él se captura el comportamiento de cada aplicación. Para capturar el comportamiento medio de cada una de ellas se seleccionan las entradas recomendadas e indicadas en la documentación de cada aplicación.

Finalizadas las fases 1 y 2, correspondientes a la caracterización de cache y el perfilado de las aplicaciones, se tienen dos tipos de resultados. De una parte, el tiempo y la energía requeridos por acceso que se obtiene para cada configuración de cache y, por otra parte, las trazas de programa con todas las operaciones de memoria de cada aplicación que se han obtenido. Por tanto, este enfoque está listo para evaluar el rendimiento de cualquier configuración de cache candidata que será generada en el módulo de optimización, basado en EA.

A continuación se explica el marco de optimización propuesto para cada uno de los algoritmos evolutivos mono-objetivo con GE y DE, y multi-objetivo con NSGA-II. La selección de estos tres algoritmos se debe, principalmente a que GE aporta gran flexibilidad a la hora de diseñar la gramática que traduce el genotipo a fenoti-

po, incluso si se cambiara el simulador de cache. Esto es debido a que el fenotipo se conforma como la llamada completa a un simulador de cache dado. Por otra parte, el problema que se aborda optimiza dos métricas, en conflicto entre sí, que son tiempo de ejecución y consumo de energía. En este sentido, la optimización multiobjetivo puede ayudar a minimizar ambas métricas de una forma independiente. Los dos enfoques anteriores abordan el problema para cada aplicación individual. La tercera propuesta propone un enfoque de optimización de forma conjunta para todas las aplicaciones seleccionadas. Así, se ha seleccionado DE por ser considerado el algoritmo más rápido de los tres. Los tres algoritmos han sido implementados utilizando la librería JECO [177], una destacada libreria de optimización disponible para su descarga pública en [178].

En primer lugar, se presenta el enfoque dirigido por GE. La metodología con GE que se propone puede ser aplicada a cualquier clase de arquitectura del procesador. En este contexto, el principal y único cambio en el marco presentado será el simulador de cache y la gramática, como se mostrará más adelante. Además si se modifican los parámetros a optimizar, el único cambio a realizar sería igualmente la definición de la gramática, que además evita que los parámetros de configuración tomen valores incorrectos. Por último, realizando una búsqueda en profundidad por la literatura, y especialmente en uno de los principales centros de publicaciones de Evolución Gramatical [179], no se han encontrado trabajos recientes que apliquen esta técnica para optimizar el diseño de cache de una microarquitectura.

5.4. Optimización de memoria cache de nivel 1 para sistemas empotrados mediante gramáticas evolutivas

Las técnicas heurísticas reducen el número de evaluaciones mientras se realiza la búsqueda hacia el óptimo, y por tanto, encajan bien en este tipo de problemas. En este proceso de optimización se presenta un esquema basado en Evolución Gramatical [180] que obtiene, usando trazas de programa, la configuración de cache optimizada en términos de tiempo de ejecución y consumo de energía, respecto a las operaciones de memoria para una aplicación dada. Comparando este esquema con la búsqueda exhaustiva, el tiempo de ejecución es muy reducido por dos razones: (1) el algoritmo metaheurístico converge incluso con un pequeño número de generaciones y tamaño de población; (2) se ha incorporado una tabla *hash* para acceder más rápidamente a los valores de la función objetivo de las configuraciones evaluadas previamente. Adicionalmente, la flexibilidad presente en la gramática en GE ayuda a generar los parámetros que el simulador de cache necesita para cada evaluación, haciendo más fácil la comunicación con el algoritmo de optimización.

En esta tesis, se propone un entorno de optimización basado en GE, que es capaz de encontrar de forma eficiente, las mejores configuraciones de cache para un conjunto de aplicaciones dado. Esta metaheurística permite una reducción importante del tiempo de optimización, a la vez que obtiene buenos resultados con un número bajo de generaciones. Además, esta reducción de tiempo se incrementa también debido a que se almacena eficientemente las memorias cache evaluadas. La apuesta por GE se debe, en gran medida, a la plasticidad que la gramática presenta para la creación de los fenotipos que forman la llamada al simulador de memorias cache. Este simulador es necesario para la evaluación de las diferentes configuraciones.

Para comprobar la propuesta realizada, se han ejecutado un conjunto de experimentos con el fin de encontrar la mejor configuración de cache para un conjunto de aplicaciones multimedia, las cuales han sido seleccionadas del conjunto de aplicaciones Mediabench. Estas aplicaciones son representativas del procesamiento de

imágenes, audio y vídeo, y son aplicaciones típicas de sistemas empotrados. En estos experimentos, se ha asumido una arquitectura hardware basada en la familia de procesadores ARM9, ampliamente utilizados en sistemas empotrados multimedia. Como se mostrará más adelante, la reducción media del tiempo de ejecución y consumo de energía de las configuraciones de cache optimizadas es de $37,75\,\%$ y $87,18\,\%$, respectivamente, en relación a una memoria cache de referencia.

Asimismo, el tiempo de ejecución de la optimización es asumible y se ha estimado una reducción media del $94\,\%$ si se compara con la implementación de la variante GE, sin almacenamiento de las evaluaciones previas de los individuos. Los resultados experimentales realizados para el subconjunto de aplicaciones Mediabench muestran que esta propuesta es capaz de encontrar configuraciones de cache que obtienen una mejora promedio del $62\,\%$ frente a una configuración real tomada como configuración cache de referencia.

La siguiente sección describe los principios de la Evolución Gramatical y cómo funciona el proceso evolutivo mediante GE para el problema propuesto.

5.4.1. Evolución gramatical

La Evolución Gramatical [181, 182, 183] es una técnica evolutiva que es capaz de representar individuos mediante gramáticas. GE es una forma de Programación Genética (GP) [126] que evoluciona programas que son evaluados conforme a su función objetivo. En este sentido, GE representa cada programa, es decir, cada individuo a través de una expresión generada mediante una gramática definida por el investigador.

GE realiza el proceso evolutivo sobre cadenas binarias de longitud variable, que se mapean para generar un programa seleccionando reglas de producción definidas en una forma *Backus-Naur* (BNF). Dado que las cadenas son cromosomas, GE permite usar operadores genéticos como selección, cruce y mutación. En el Capítulo 3, en la Sección 3.2.5 se realiza una introducción más amplia de esta técnica.

De esta forma, el espacio de búsqueda de las configuraciones de cache es re-

corrido por la GE de acuerdo a la definición de la gramática. Para cada individuo, se llama al simulador de cache para obtener el número de aciertos y fallos para las aplicaciones objetivo. A continuación, utilizando el tiempo de acceso y la energía consumida por acceso, anteriormente calculadas mediante CACTI, y los modelos de rendimiento y de energía descritos en la sección 5.3.1, se calcula la función objetivo que se describe más adelante. Este proceso se repite hasta alcanzar el número de generaciones seleccionados para GE. Como resultado, se obtiene una configuración de cache para cada uno de los programas seleccionados.

El espacio de búsqueda de este problema se define por el conjunto de configuraciones de cache que se pueden formar por el conjunto de parámetros completos mostrado en la Figura 5.1 de la sección 5.3.3. Sin embargo, GE trabaja con individuos que representan aquellas configuraciones bajo una codificación apropiada. De hecho, GE utiliza un genoma lineal en lugar de una estructura basada en árbol, como sucede en GP. El genoma lineal facilita la aplicación de operadores genéticos. Aquí, el cromosoma de cada individuo consiste en una cadena de valores enteros que se corresponden con los codones. Cada codón almacena un valor de 8 bits que puede ser ampliado, ya que generalmente almacena un valor módulo número máximo de alternativas, y forma el genotipo. El genotipo se decodifica en un proceso denominado mapeo, donde se utiliza una gramática para convertir los codones en el fenotipo del individuo.

Un individuo representa una configuración de cache que debe ser evaluada para medir su tiempo de ejecución y consumo de energía. En este sentido, se ha decidido considerar el mismo peso para ambas características de cache. Así, para una configuración de cache dada c_i, el proceso de evaluación es conducido por la función objetivo f definida en la Ecuación (5.4).

$$f(c_i) = 0{,}5 \times \frac{T(c_i)}{T(Baseline)} + 0{,}5 \times \frac{E(c_i)}{E(Baseline)} \tag{5.4}$$

Cada componente de la función objetivo, es decir, tiempo de ejecución y consumo de energía, se normalizan con respecto a una configuración base. Una configuración base es una configuración de referencia con la que comparar los resultados

de cada configuración de cache a evaluar. Al normalizar a una configuración de base, se es capaz de escalar correctamente y operar con las unidades de las diferentes medidas como el tiempo y la energía. Como se muestra en la Ecuación (5.4), ambos términos se ponderan al 50 % buscando un mejor equilibrio entre la mejora del rendimiento y el consumo de energía. La mejor configuración de cache en GE será aquella que minimice el valor objetivo.

Dado que la evaluación de una configuración de cache implica una llamada al simulador de cache, se ha diseñado una gramática que produce fenotipos que forman los parámetros necesarios para realizar la llamada al simulador. De hecho, uno de los beneficios de GE es la habilidad para crear fenotipos complejos que vienen de un genotipo formado por valores enteros.

La Figura 5.3 muestra, en formato BNF, la gramática que se ha diseñado para la optimización de cache bajo el espacio de búsqueda anteriormente definido. Como se ha mencionado anteriormente, la gramática produce la cadena de parámetros que será enviada al simulador de cache. De hecho, se adaptan los parámetros a los requisitos del simulador. Como se muestra en la figura, el algoritmo de reemplazo (regla IV), tomará valores como `l`, `f` o `r`, que corresponden en el simulador, respectivamente, a `LRU`, `FIFO` y `RANDOM`. La misma metodología se aplica para el algoritmo de prebúsqueda y la política de escritura.

Como se explica en el Capítulo 3, una gramática se describe como una tupla {*N, T, S, P*}, donde *N* representa los símbolos no terminales, *T* es el conjunto de terminales, *S* representa el símbolo inicial perteneciente a *N*, y *P* define el conjunto de reglas de producción que mapea los valores de *N* a *T*. El símbolo "|" separa las diferentes opciones dentro de una regla de producción. En la gramática diseñada, *P* está formada por las reglas de producción definidas de `I` hasta `VII`.

A continuación, se muestra un ejemplo del proceso de mapeo para el problema que se está tratando, usando la gramática definida en Figura 5.3. Se considera el genotipo mostrado en la parte superior de la Figura 5.4 como una solución candidata. Este individuo es un genotipo que representa una configuración de cache. Sin

```
N = { <DineroParams>, <CacheSizeB>, <LineSizeB>,
     <ReplAlg>, <Assoc>, <PrefAlg>, <WritePol> }
T = { 1, 2, 4, 8, 16, 32, 64, 128 512, 1024, 2048,
     4096, 8192, 16384, 32768, 65536, l, f, r, m,
     d, a, n }
S = <DineroParams>

P = { I <DineroParams> ::= -l1-isize <CacheSizeB>
                           -l1-ibsize <LineSizeB>
                           -l1-irepl <ReplAlg>
                           -l1-iassoc <Assoc>
                           -l1-ifetch <PrefAlg>
                           -l1-dsize <CacheSizeB>
                           -l1-dbsize <LineSizeB>
                           -l1-drepl <ReplAlg>
                           -l1-dassoc <Assoc>
                           -l1-dfetch <PrefAlg>
                           -l1-dwback <WritePol>

     II <CacheSizeB> ::= 512 | 1024 | 2048 | 4096
                       | 8192 | 16384 | 32768
                       | 65536
     III <LineSizeB> ::= 8 | 16 | 32 | 64
     IV <ReplAlg> ::= l | f | r
     V <Assoc> ::= 1 | 2 | 4 | 8 | 16 | 32 | 64
                 | 128
     VI <PrefAlg> ::= m | d | a
     VII <WritePol> ::= a | n }
```

Figura 5.3: Gramática para la descripción de la configuración de cache. N contiene los símbolos no terminales; T representa los símbolos terminales; S es el símbolo inicial y P contiene las diferentes reglas de producción. Se comienza con la sustitución del símbolo inicial por su regla de producción, que es la llamada al simulador de cache con sus parámetros correspondientes como símbolos no terminales y terminales.

embargo, tiene que ser decodificado para obtener los valores de cada uno de los 11 parámetros de la configuración de cache descritos anteriormente.

El proceso de mapeo o decodificación se realiza aplicando la operación módulo entre el valor del codón y el número de opciones de cada regla de producción correspondiente al símbolo no terminal que está siendo procesado.

A continuación, considerando el individuo de ejemplo, el proceso de decodificación se realiza como se muestra en la Figura 5.4. Comienza con la decodificación del símbolo inicial <DineroParams>, que se sustituye por su producción, que corresponde a la llamada parametrizada al simulador de cache. En este punto, están presentes varios símbolos terminales y el primero de ellos es <CacheSizeB>. Por tanto, se selecciona el primer gen del genotipo, 20. Dado que la regla de producción para <CacheSizeB> (regla II), tiene 8 opciones diferentes, el valor seleccionado es la opción 4 (20 MOD 8 = 4), que corresponde a 8192. Se observa que la primera

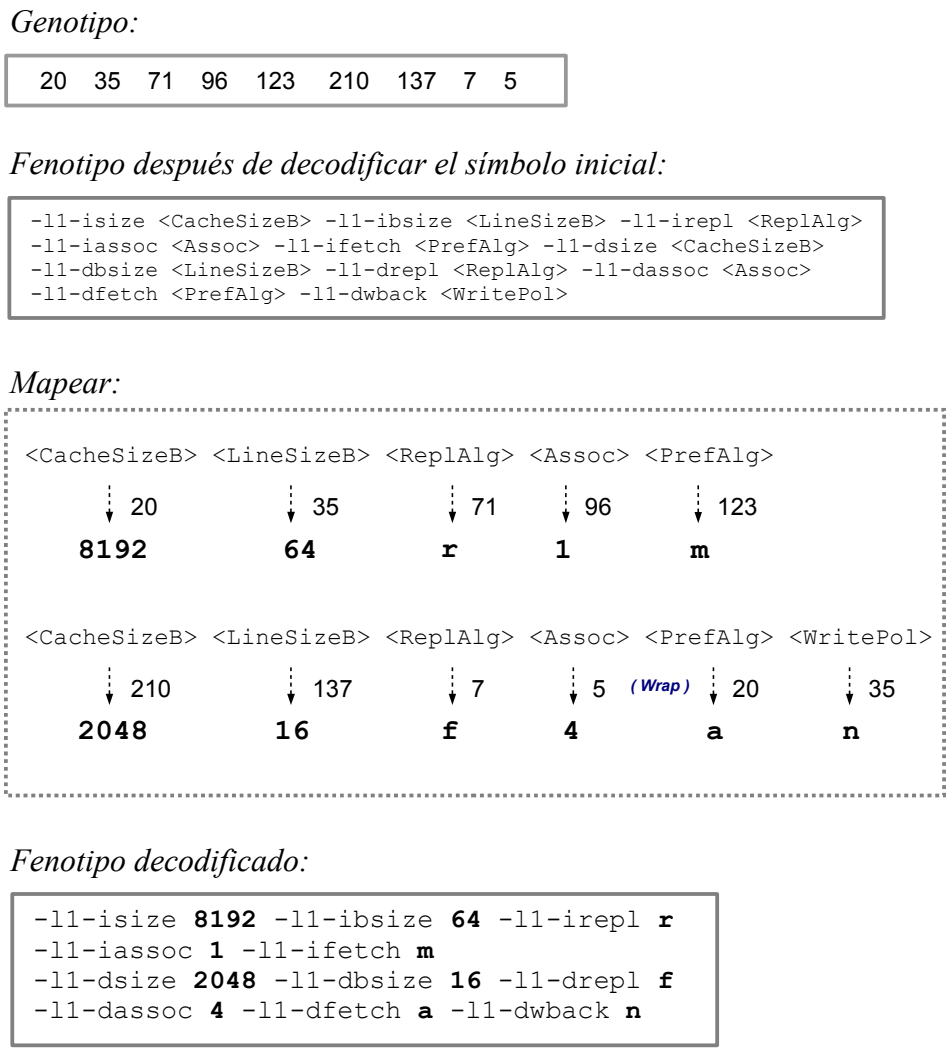


Figura 5.4: Proceso de decodificación en GE que, comenzando desde el genotipo superior, obtiene el fenotipo mediante mapeo a través de la gramática propuesta.

opción de la regla está unida a 0, la segunda a 1 y así sucesivamente. Seguidamente, se lee el segundo gen y su valor, 35, se utiliza para decodificar el siguiente símbolo no terminal, <LineSizeB>. La regla de producción para este símbolo es III, y tiene 4 opciones. Entonces, después de la operación módulo (35 MOD 4 = 3) <LineSizeB> se reemplaza por 64.

El siguiente símbolo no terminal a decodificarse es <ReplAlg>, cuya regla de producción, IV tiene 3 opciones diferentes. Así, se procesa el valor del codón 71. Dado que 71 MOD 3 = 2, el valor seleccionado es r. De forma análoga, el siguiente codón, 96 para <Assoc> se convierte en 1 dado que 96 MOD 8 = 0. El último símbolo no terminal correspondiente a los parámetros de la cache de instrucciones,

<PrefAlg>, se determina por el valor del codón 123 que se reemplaza por m, una vez que 123 MOD 3 = 0. A continuación, el proceso de mapeo continúa con el codón 210 y el segundo <CacheSizeB> símbolo, que mapea 2048 correspondiente al tamaño de la cache de datos. Después, 137 mapea <LineSizeB> a 16, el gen 7 mapea <ReplAlg> a f y 5 mapea <Assoc> a 4.

En este punto, todos los genes han sido utilizados pero la expresión está aún parcialmente decodificada. Sin embargo, GE es capaz de trabajar con cromosomas de longitud variable, y si el proceso de decodificación llega al final del cromosoma y aún hay símbolos no terminales, GE va al primer codón y permite la decodificación hasta el final. Este proceso se denomina *wrapping*. Por tanto, en nuestro ejemplo se produce un *wrap* en este punto. De esta forma, <PrefAlg> se decodifica con el primer gen, 20, obteniendo a. Para finalizar, el proceso de decodificación finaliza mapeando <WritePol> a n con el gen cuyo valor es 35.

Como resultado, nuestro ejemplo de genotipo devuelve el fenotipo mostrado en la parte baja de la Figura 5.4 que representa una configuración de cache, donde el tamaño de cache es 8192, el tamaño de bloque de la cache de instrucciones es 64, y así sucesivamente.

Una vez decodificado, el fenotipo se usa para llamar al simulador de cache y evaluar la configuración correspondiente en la aplicación objetivo, obteniendo el tiempo de ejecución y el consumo de energía de acuerdo al valor objetivo definido en la Ecuación (5.4). Al finalizar el número de generaciones especificadas en el algoritmo, GE devuelve la configuración de cache que obtiene el mejor valor objetivo.

Como se observa, la gramática determina tanto los parámetros concretos a ser optimizados como la comunicación con el simulador de memorias cache, que proporciona el valor de la función objetivo. Así, cambiar la gramática permitirá adoptar diferentes enfoques de optimización: fijar el valor de algunos parámetros, añadir nuevos parámetros, cambiar el simulador de cache, usar argumentos de longitud variable en el simulador de cache (para algoritmos personalizados, por ejemplo), y

así sucesivamente. Todo ello sin modificar el código fuente del algoritmo de optimización. Por otro lado, alguna metodología de optimización basada en un número fijo de variables como, por ejemplo un GA, requerirá la modificación de una parte importante del código, principalmente en los módulos de decodificación y evaluación del valor objetivo. Por tanto, este enfoque basado en GE proporciona una gran flexibilidad, así como extensibilidad.

5.4.2. Resultados experimentales

Con el fin de probar el marco experimental, se han seleccionado doce aplicaciones del conjunto Mediabench, que son representativas de las aplicaciones multimedia en sistemas empotrados. A continuación, se describen las características generales de estas aplicaciones. La descripción completa se puede encontrar en la página web de Mediabench [184].

- JPEG es un software desarrollado en C para comprimir (*cjpeg*) y descomprimir (*djpeg*) imágenes JPEG en color y escala de grises.
- MPEG es una implementación optimizada para transmitir vídeo digital de alta calidad. La aplicación *mpegenc* se encarga del proceso de codificación, y la aplicación *mpegdec* del proceso de decodificación.
- GSM implementa un algoritmo para reconocimiento seguro de voz. El algoritmo se basa en la implementación del estándar provisional Europeo GSM 06.10 para transcodificación de voz a máxima velocidad. La aplicación *gsmenc* realiza el proceso de compresión de fotogramas, y *gsmdec* el proceso de descompresión.
- *Pegwit* implementa el proceso para cifrado de clave pública y autenticación. El proceso utiliza una curva elíptica sobre GF(2^{255}), el algoritmo de codificación SHA1 para hash, y el cifrado simétrico por bloques cuadrado. *Pegwitenc* y *pegwitdec* se encargan de la codificación y decodificación, respectivamente.
- EPIC (Efficient Pyramid Image Coder) se encarga de la compresión de datos

de imagen (*epic*) y del proceso de descompresión (*unepic*). Los algoritmos de compresión están basados en una descomposición biortogonal de onda diádica críticamente muestreada y un codificador de entropía run-length/Huffman combinado.

- ADPCM (Adaptive Differential Pulse Code Modulation) implementa los algoritmos de compresión y descompresión de voz. *Rawcaudio* realiza la compresión, mientras que *rawdaudio* se encarga de la descompresión. El algoritmo usado consigue una tasa de compresión de 4:1 en muestras PCM lineales de 16 bits.

Los experimentos se han realizado en una máquina provista con un procesador Intel i5 660 a 3.3 GHz. Dado que este es un procesador de 4 cores, se ejecutan 4 experimentos de forma simultánea. La máquina dispone de 8GB de RAM y el sistema operativo ha sido Ubuntu Desktop 14.04.

Diseño de experimentos

Como se ha mencionado anteriormente, el marco experimental se personaliza de acuerdo a la arquitectura ARM. A continuación, para evaluar el enfoque propuesto se considera un primer nivel de memoria cache con un rango de tamaños entre 512B y 64KB, con incrementos potencia de 2. Con el conjunto de tamaños de cache definidos se cubren muchos de los tamaños soportados por algunos de los dispositivos de la familia de procesadores ARM9. Sin embargo, la caracterización de cache se realiza para memorias individuales teniendo en cuenta sólo los parámetros hardware en el proceso de caracterización *off-line*. En el espacio de búsqueda, descrito en la sección 5.3.3, estos parámetros se corresponden con 8 tamaños de cache posibles, 8 grados diferentes de asociatividad y 4 valores posibles de tamaño de bloque, como se muestra en la Figura 5.1. Por tanto, el número de caracterizaciones de cache a ejecutar son 256, proceso que, además, sólo se realiza una única vez.

Usando esta metodología para la obtención de perfiles *off-line*, se han generado las trazas de las aplicaciones Mediabench usando las herramientas Trimaran, co-

nectado con SimpleScalar para evaluar la arquitectura ARM. Teniendo en cuenta lo anterior, se han realizado los cambios adecuados para las herramientas SimpleScalar y Trimaran para generar las trazas de las aplicaciones que pueden ser procesadas por el simulador de cache Dinero IV.

Este tipo de experimentos necesita gestionar las trazas completas de las aplicaciones o ejecutar un número apropiado de instrucciones que reflejen el comportamiento del programa completo. Así se evita capturar un comportamiento parcial de una fase determinada. Por ejemplo, un comportamiento de fase tiene lugar al inicio de la ejecución de la aplicación, cuando todos los bloques de datos se deben copiar desde la memoria principal, por tanto todos los accesos a la memoria cache son fallos. Así, un pequeño número de instrucciones pueden proporcionar resultados que no pueden ser considerados como significativos o concluyentes. En este sentido, cada aplicación se ha simulado un máximo de $7{,}5 \times 10^7$ instrucciones, o incluso la aplicación completa, si el número total de instrucciones para la ejecución completa es menor que este valor. El objetivo es alcanzar un equilibrio entre el tiempo de simulación, el tamaño de las trazas de programa y un número adecuado de instrucciones.

Tabla 5.2: Configuración para GE.

Número de generaciones	100
Tamaño de población	50
Probabilidad de cruce	$0,9$
Probabilidad de mutación	$1/11$

Adicionalmente, el algoritmo GE se ha ejecutado 10 veces para cada aplicación objetivo con el fin de reducir la probabilidad de alcanzar un óptimo local.

La Tabla 5.2 muestra la configuración para GE. Los operadores que se han aplicado han sido selección por torneo con dominancia simple, cruce de un único punto y el operador de mutación de enteros clásico (*integer-flip*) con probabilidad según las recomendaciones en [150].

Tabla 5.3: Configuración base de cache. Se forma con los parámetros de la cache de instrucciones y de datos.

Cache de Instrucciones				
Tamaño	**Tamaño de Bloque.**	**Asociatividad**	**Alg. Reemplazo**	**Alg. Búsqueda**
16 KB	32 B	4	LRU	on-demand

Cache de Datos					
Tamaño	**Tamaño de Bloque.**	**Asociatividad**	**Alg. Reemplazo**	**Alg. Búsqueda**	**Pol. Escritura**
16 KB	32 B	4	LRU	on-demand	Copy-Back

Como se ha mencionado en la Ecuación (5.4) (ver sección 5.4.1), el algoritmo necesita una configuración base de cache para calcular el valor objetivo de un individuo. Para estos experimentos, se ha seleccionado una configuración base de cache que es similar a la cache del primer núcleo en la consola portátil de videojuegos GP2X, que dispone de un procesador dual ARM940T a 200 MHz.

La Tabla 5.3 muestra el tamaño de cache, tamaño de bloque, número de vías o grado de asociatividad, política de reemplazo, algoritmo de búsqueda y, para la cache de datos, la política de escritura de la configuración base.

El tiempo de ejecución y consumo de energía para la configuración base de cache se ha calculado siguiendo los modelos descritos en la sección 5.3.1.

5.4.3. Función objetivo

En primer lugar, los resultados se analizan de acuerdo al valor de la función objetivo. Se recuerda que el valor objetivo 1 corresponde al peor valor, es decir, al tiempo de ejecución y al consumo de energía de la configuración base de cache ejecutando cada una de las aplicaciones. Por lo tanto, se minimiza la función objetivo para reducir ambas características.

La Tabla 5.4 muestra las siguientes estadísticas en valor promedio para cada una de las aplicaciones: valor objetivo, porcentaje de desviación con respecto al mejor valor objetivo encontrado y el número de soluciones que alcanzan dicho valor.

Como se aprecia en esta tabla, los valores objetivo promedio oscilan entre 0,47 y 0,30, lo cual significa mejoras entre el 53 % y 70 %, con una mejora promedio

Tabla 5.4: Valores de la función objetivo

Aplicación	Avg. Obj.	Dev. (%)	# Mejor
cjpeg	0,3986650026	0,74 %	8
djpeg	0,4746031647	3,06 %	1
epic	0,3365548961	1,24 %	7
unepic	0,3977187056	0,65 %	1
gsmdec	0,3857290316	1,15 %	7
gsmenc	0,4442631856	1,82 %	4
mpegdec	0,3682433302	1,09 %	6
mpegenc	0,3631493163	0,89 %	3
pegwitdec	0,3663486721	0,01 %	9
pegwitenc	0,3568943181	0,00 %	10
rawcaudio	0,3052829963	0,00 %	8
rawdaudio	0,31533024	0,56 %	9

del 62 % en el valor objetivo. Por tanto, la optimización obtiene mejoras importantes del objetivo. Adicionalmente, los bajos valores de la desviación típica muestran que GE obtiene valores muy cercanos a lo largo de las ejecuciones del proceso de optimización. De hecho, el número de veces que se encuentra el mejor resultado es mayor o igual al 70 % en 7 de las 12 aplicaciones. Particularmente, GE obtiene excelentes resultados en las aplicaciones *pegwitdec*, *pegwitenc*, *rawcaudio* y *rawdaudio*. Por otra parte, a pesar de que GE obtiene el mejor valor sólo una vez en *djpeg* y *unepic*, la desviación está por debajo del 3,1 % y 0,7 %, respectivamente, lo cual es, de nuevo, un resultado positivo.

Estos resultados demuestran que una metaheurística como GE, incluso con un número bajo de generaciones y tamaño de población, es capaz de encontrar soluciones donde la función objetivo se reduce por encima del 62 % para el problema de optimización de la memoria cache.

Tiempo de ejecución de la optimización

Se ha realizado un análisis adicional con respecto al tiempo de ejecución del proceso de optimización (RT, por sus siglas en inglés *Run Time*). La Tabla 5.5 muestra

el tiempo de ejecución promedio del proceso de optimización en horas, el número medio de evaluaciones en relación al máximo previsto para GE, el tiempo promedio para la evaluación de cada una de las configuraciones de cache en segundos, el máximo tiempo de ejecución de la optimización de GE, si se hubieran realizado todas las evaluaciones, y el ahorro en tiempo de ejecución de los experimentos en relación a ese tiempo, gracias a que se memorizan los individuos ya evaluados.

Tabla 5.5: Valores de tiempo de ejecución. Para cada aplicación, se muestra tiempo de ejecución promedio (horas), número medio de evaluaciones vs. max. GE, tiempo de ejecución promedio para cada evaluación (segundos), máximo tiempo de ejecución para GE (sin guardar evaluaciones anteriores) y ahorro en tiempo de ejecución en relación al máximo previsto para GE.

Aplicación	RT (h.)	# Eval. (%)	Eval. (seg.)	Max GE (h.)	Ahorro RT
cjpeg	2,185	5,96 %	26,200	36,389	93,99 %
djpeg	2,647	5,74 %	33,064	45,923	94,24 %
epic	1,405	5,87 %	17,148	23,816	94,10 %
unepic	0,350	5,79 %	4,340	6,028	94,19 %
gsmdec	0,004	6,33 %	0,043	0,060	93,67 %
gsmenc	0,003	5,58 %	0,041	0,057	94,41 %
mpegdec	2,275	5,69 %	28,805	40,006	94,31 %
mpegenc	2,031	5,80 %	25,209	35,013	94,20 %
pegwitdec	0,945	6,38 %	10,667	14,815	93,62 %
pegwitenc	0,869	5,98 %	10,466	14,537	94,02 %
rawcaudio	0,372	5,95 %	4,483	6,227	94,03 %
rawdaudio	0,274	5,89 %	3,343	4,642	94,10 %

Mientras el proceso de optimización se ejecuta, se han tomado medidas del tiempo necesario para evaluar cada configuración de cache. Como se observa en la tabla, el tiempo de evaluación medio es alto en alguna de las aplicaciones como *djpeg*, *mpegdec* y *mpegenc*. Teniendo en cuenta estos valores de tiempo, se ha calculado el tiempo que consumiría GE si se realizaran 5000 evaluaciones, es decir, si el algoritmo no memorizara el resultado de los individuos ya evaluados.

Como se puede observar en la columna “Ahorro RT”, el tiempo de ejecución se reduce en más de un 93,6 %. Por tanto, es evidente que la GE implementada es

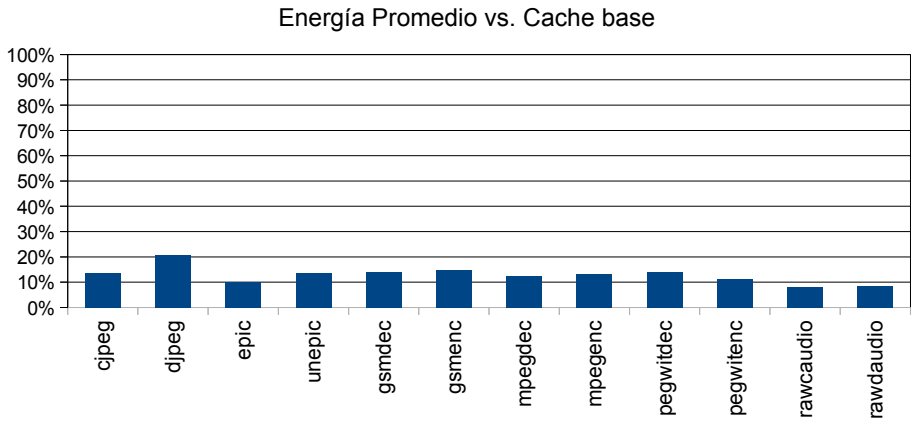


Figura 5.5: Consumo de energía promedio para las soluciones obtenidas en cada aplicación en relación a la configuración base (100 %).

capaz de realizar las ejecuciones en un tiempo razonable debido a que memoriza aquellos individuos que ya han sido evaluados.

Los reducidos tiempos de ejecución se deben a dos factores. Por una parte, el número de evaluaciones es bajo porque la exploración del algoritmo alcanza un área de valores mínimos que hacen que la población converja a ese mínimo. En otras palabras, debido a que no tenemos medida de diversidad, es probable que los individuos de la población sean parecidos en las diferentes generaciones. Por otra parte, dado que el algoritmo recuerda la evaluación de cada individuo, las configuraciones evaluadas previamente no se envían al simulador externo, sino que se leen de la memoria que mantiene GE. Este es un proceso muy rápido porque se ha implementado mediante tablas *hash*.

Rendimiento de las soluciones

Como se indica en la sección 5.4.1, la función objetivo incorpora los valores de tiempo de ejecución y consumo de energía para una configuración de cache dada. Una vez que se han obtenido la configuraciones de cache optimizadas, se muestra la interpretación de estos resultados en términos de tiempo de ejecución y consumo de energía.

La Figura 5.5 describe el consumo promedio de energía de las configuraciones de cache de cada aplicación, considerando que el 100 % es el consumo de energía

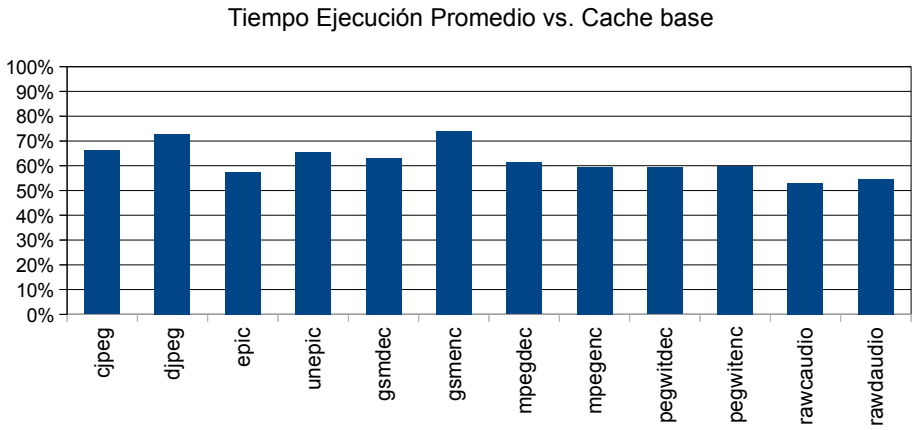


Figura 5.6: Tiempo de ejecución promedio para las soluciones obtenidas en cada aplicación en relación a la configuración base (100 %).

de la configuración base de cache. De media, las soluciones obtenidas alcanzan el 12, 82 % de la configuración base.

La Figura 5.6 muestra el tiempo promedio de la ejecución de las configuraciones de cache de cada aplicación, considerando que el 100 % es el consumo de energía de la configuración cache de base. De media, las soluciones que se han obtenido alcanzan el 62, 25 % de la configuración base.

En resumen, se ha mejorado el comportamiento de la memoria cache en un 87, 18 % el consumo de energía y, al mismo tiempo, en un 37, 75 % el tiempo de ejecución considerando una configuración base comercial. Estos resultados se han obtenido en un tiempo muy corto comparado con el tiempo esperado para una GE que no recuerde evaluaciones previas de un mismo individuo.

Por razones de claridad, se destaca que la configuración base actúa como un punto de referencia desde el cual optimiza el proceso GE. De hecho, dado que este marco de trabajo utiliza la caracterización desde un emulador de cache (CACTI en este caso), cualquier configuración de base podría optimizarse proporcionando el marco con los correspondientes valores de caracterización de la configuración base.

5.4.4. Conclusiones

El primer enfoque dentro del marco de optimización presentado es dirigido por GE y se divide en 3 fases. Las dos primeras fases se realizan *off-line* y son respon-

sables de suministrar las trazas de programa y las características del conjunto de configuraciones hardware posibles para las memorias cache. La tercera fase es un proceso de optimización que aplica GE para obtener la configuración que podría obtener el mejor comportamiento para cada aplicación objetivo. Para evaluar una configuración cache, GE obtiene el rendimiento de cada configuración con la ayuda del simulador de cache Dinero IV. En este sentido, la gramática de GE facilita la comunicación con Dinero IV dado que el fenotipo se forma mediante los parámetros y formatos requeridos por la llamada al simulador, dirigido por la gramática.

Adicionalmente, este enfoque permite modificar el número de parámetros a optimizar, así como los cambios del simulador de cache, simplemente modificando la gramática que nos ocupa. Además, si el simulador de cache permitiera parámetros de longitud variable, una gramática apropiada sería capaz de generar la llamada correcta al simulador sin ninguna modificación en el marco de optimización.

Este propuesta ha sido probada con 12 aplicaciones pertenecientes al conjunto de aplicaciones Mediabench considerando una configuración cache real como configuración base. Los resultados han obtenido una mejora promedio del $87,18\,\%$ en el consumo de energía y $37,75\,\%$ en tiempo de ejecución frente a la configuración base de cache.

La simplificación de los procesos o el diseño de un marco efectivo que permita personalizar el proceso de optimización para un conjunto de aplicaciones objetivo a la vez son modificaciones que mejorarían el proceso propuesto. También se debería tener en cuenta las diferentes cargas que las operaciones de memoria representan en las aplicaciones objetivo para equilibrar de forma precisa su peso en la optimización. De hecho, un paso natural a seguir después de esta investigación será analizar la influencia entre los resultados de tiempo de ejecución y energía, y considerar optimización multiobjetivo, tal y como se hace en el siguiente apartado.

5.5. Optimización multi-objetivo del consumo de energía y tiempo de ejecución en una memoria cache de nivel 1 para sistemas empotrados

En esta sección se presenta el enfoque dirigido por un algoritmo multi-objetivo como técnica evolutiva, dentro del marco de optimización implementado. Siendo las métricas a optimizar rendimiento y consumo de energía, en conflicto entre sí, la optimización multi-objetivo puede ayudar a minimizar ambas métricas, de una forma independiente. En esta tesis, se propone un método de optimización, basado en algoritmos evolutivos multi-objetivo, que es capaz de obtener una configuración de cache optimizada para un conjunto de aplicaciones Mediabench. Con el fin de contrastar los resultados, estos se comparan con una memoria cache base, sobre la que se alcanza una mejora promedio del 64, 43 % y 91, 69 % en tiempo de ejecución y consumo de energía, respectivamente.

Por tanto, en este capítulo de tesis se presenta una nueva metodología para evaluar las configuraciones de cache y personalizar los diseños de cache con el objetivo de reducir tanto el tiempo de ejecución como el consumo de energía mediante optimización multi-objetivo [143]. La optimización multi-objetivo permite obtener soluciones donde se alcanzan valores aceptables para ambos valores objetivos simultáneamente. Particularmente, el marco de optimización se construye sobre el algoritmo NSGA-II. Para evaluar este enfoque, se han diseñado automáticamente caches optimizadas para un conjunto de aplicaciones multimedia del conjunto de aplicaciones Mediabench, dada su representatividad para el procesamiento de imagen, audio y vídeo. La arquitectura hardware seleccionada se basa en el procesador ARM920T [70], ampliamente utilizado en dispositivos empotrados multimedia.

El objetivo es encontrar la mejor configuración de cache que minimice el tiempo de acceso a memoria (rendimiento) y el consumo de energía. Dado que se intentan minimizar dos objetivos contradictorios, la optimización multi-objetivo es adecuada para abordar este problema. Este enfoque, si bien puede ser aplicado a cualquier tipo de sistema, es especialmente válido en sistemas empotrados, donde el número

pequeño de aplicaciones permite al ingeniero seleccionar un diseño de cache entre todas las optimizaciones realizadas, como se muestra en esta tesis.

5.5.1. Representación del espacio de búsqueda

En este marco de optimización multi-objetivo, se ha utilizado una parte del espacio de búsqueda completo mostrado en la Figura 5.1 en la sección 5.3.3. Por tanto, aunque esta metodología permite definir varios tamaños de cache, en una primera fase del enfoque que aquí se presenta sólo se incluye uno, que es el valor por defecto de la memoria cache del procesador ARM920T, el sistema objetivo. Por tanto, se considera un tamaño de cache fijo de 16KB. Además, se establecen únicamente cinco valores posibles para el parámetro de asociatividad (4, 8, 16, 32 y 64 vías). El resto de parámetros mantienen el mismo número de valores posibles especificados en la Figura 5.1.

Teniendo en cuenta los parámetros seleccionados y los valores posibles para cada uno de ellos, el tamaño del espacio de búsqueda es de 64800 configuraciones de cache para cada tamaño de cache. De esta forma, una técnica determinista puede tardar mucho tiempo en encontrar una solución óptima como se muestra en la sección 5.6.3, dado que cada configuración debe ser evaluada con una traza de programa. Así, las técnicas heurísticas encajan bien en la solución de problemas de optimización multi-objetivo, especialmente cuando se debe minimizar un conjunto de objetivos de diseño que entran en conflicto. En este sentido, los MOEAs normalmente proporcionan buenos resultados en un entorno multi-objetivo. Concretamente, se propone el uso del algoritmo NSGA-II. De acuerdo a esta selección, es necesario codificar los valores de los diferentes parámetros para el desarrollo adecuado de la técnica seleccionada.

El cromosoma depende de la clase de problema a solucionar y su correcta codificación es fundamental para obtener la solución óptima. Este enfoque trabaja con la cache de instrucciones y la cache de datos del primer nivel de cache, y ambas deben ser configuradas con valores apropiados para cada parámetro. Así, una posi-

ble solución o individuo se define como una configuración de cache concreta para I-cache y D-cache. De esta forma, los genes de los individuos se refieren a los posibles valores de los parámetros de cache. Por tanto, un cromosoma se define como la secuencia de parámetros para una configuración de cache determinada, donde cada gen se codifica como un valor entero. En este enfoque un cromosoma podría parecerse al que aparece en la Figura 5.7.

Cache de Instrucciones				Cache de Datos				
Tam. Bloque	Asociatividad	Alg. Reemplazo	Alg. Prebúsqueda	Tam. Bloque	Asociatividad	Alg. Reemplazo	Alg. Prebúsqueda	Pol. Escritura

Figura 5.7: Especificación del cromosoma para una configuración de cache.

El cromosoma aplica un esquema de codificación donde cada gen se representa como un valor entero que se mapea a los símbolos alfanuméricos (por ejemplo, 8, 16, 32 para tamaño de bloque y LRU, FIFO, RANDOM para política de reemplazo), definidos en la Figura 5.1 y donde se consideran los valores de izquierda a derecha, comenzando en 0.

Como ejemplo para explicar el esquema de codificación, se considera el cromosoma de la Figura 5.8, el cual será decodificado según se establece en la Figura 5.1. El primer y quinto gen, “1” y “0”, se corresponden con el tamaño de bloque para I-cache y D-cache, que se mapean a 16 y 8 bytes, respectivamente. A continuación, el segundo y sexto gen, “0” and “2”, correspondientes al grado de asociatividad para I-cache y D-cache, se mapean a 4 y 16 vías. Los genes tercero y séptimo, “1” y “0”, representan el algoritmo de reemplazo y son mapeados a los algoritmos FIFO y LRU. Los genes cuarto y octavo con los valores “2” y “0” están asociados al algoritmo de prebúsqueda y se mapean a *miss-prefetch* y *on-demand*. Finalmente, el noveno gen con el valor “0” es mapeado a la política de escritura copy-back, para la cache de datos. Así, el genoma completamente decodificado se puede observar en la Figura 5.9.

Cache de Instrucciones				Cache de Datos				
1	0	1	2	0	2	0	0	0

Figura 5.8: Cromosoma o genoma de un individuo.

Cache de Instrucciones				Cache de Datos				
16 B	4 vías	FIFO	MISS-PREFETCH	8 B	16 vías	LRU	COPY-BACK	ON-DEMAND

Figura 5.9: Genoma decodificado de un individuo.

5.5.2. Función multi-objetivo

El enfoque que se propone en esta tesis define una variable de decisión $\mathbf{x} \in \mathbf{X}$ en un contexto de optimización multi-objetivo (consultar el capítulo 3, sección 3.3). La variable $\mathbf{x}$ define un conjunto de valores de los parámetros de cache que representan una configuración de cache a evaluar. El proceso de evaluación consiste en calcular la función multi-objetivo $\mathbf{f}$ como el tiempo de ejecución y el consumo de energía, ambos relativos a las operaciones en la memoria cache. Por lo tanto, la mejor configuración de cache corresponderá a aquella que obtenga los valores más bajos de tiempo de ejecución y consumo de energía. Ambos objetivos contradictorios, tal y como se ha mencionado anteriormente.

Para evaluar las diferentes configuraciones de cache, se han aplicado los modelos de energía y rendimiento basados en [74], explicados previamente en la sección 5.3.1. El diseño de la arquitectura del sistema empotrado objetivo consiste en un procesador con un nivel de cache que posee una cache de instrucciones, una cache de datos y una DRAM empotrada como memoria principal. Tanto la cache de instrucciones como la de datos se han configurado con un tamaño de 16 KB, de acuerdo a las características del procesador ARM920T, la plataforma objetivo. La cache de instrucciones es de sólo lectura y la memoria principal tiene un tamaño de 64 MB según las especificaciones de dispositivos como, por ejemplo, el navegador de coche HS-3502.

Componentes de la función multi-objetivo

El primer objetivo se corresponde con el modelo de rendimiento que permite calcular el tiempo de ejecución, tal y como se describe en la sección 5.3.1, y que se calcula mediante la Ecuación (5.1). El tiempo de ejecución se calcula según el tiempo necesario para resolver los accesos y fallos a la memoria cache. El segundo objetivo corresponde al modelo de energía, descrito también en la sección 5.3.1, y dirigido por la Ecuación (5.2).

El proceso de optimización ejecuta el algoritmo que evoluciona para minimizar el tiempo de ejecución y el consumo de energía. Después del número de generaciones estipulados, el algoritmo devuelve un frente de Pareto o aproximación al POF, que representa el mejor conjunto de configuraciones para aplicar a la memoria cache. A mayor número de generaciones, mejor es la calidad de la memoria cache.

5.5.3. Marco de optimización

En la sección 5.6.2, se describe la metodología general utilizada para optimizar las memorias caches para sistemas empotrados multimedia. La Figura 5.2 describe todos los pasos necesarios para llevar a cabo el proceso de optimización. Los procesos etiquetados como 1 (caracterización de cache) y 2 (perfilado de aplicaciones), se han explicado detalladamente en la sección 5.3.3, dado que son comunes en los tres algoritmos evolutivos propuestos. En esta sección, se detallan las particularidades que afectan a la fase 3 que corresponde al algoritmo de optimización.

La tercera fase es la *optimización de cache*. Esta fase debe ser repetida para cada aplicación objetivo y se lleva a cabo por el algoritmo NSGA-II implementado en la librería JECO. NSGA-II evalúa cada solución candidata llamando a Dinero IV. Este simulador conducido por trazas recibe una configuración de cache de NSGA-II y devuelve el número de aciertos y fallos de cache para la traza correspondiente de la aplicación objetivo. Estos datos, junto a los parámetros obtenidos en la primera fase se incluyen en la función multi-objetivo para calcular tanto el tiempo de ejecución como la energía consumida.

Se ha seleccionado NSGA-II como algoritmo de optimización multi-objetivo porque, según estudios publicados recientemente en [185], el algoritmo evolutivo actual de facto para optimización multi-objetivo es NSGA-II. Este estudio indica que NSGA-II se ha utilizado como único algoritmo en el 53 % de los artículos examinados, posicionando el algoritmo como uno de los MOEAs más ampliamente utilizado, y obteniendo resultados muy competitivos. Dado que el objetivo de esta tesis es proporcionar una técnica que automáticamente diseñe configuraciones de memorias cache optimizadas, y no buscar el mejor algoritmo de optimización, se propone el uso de NSGA-II.

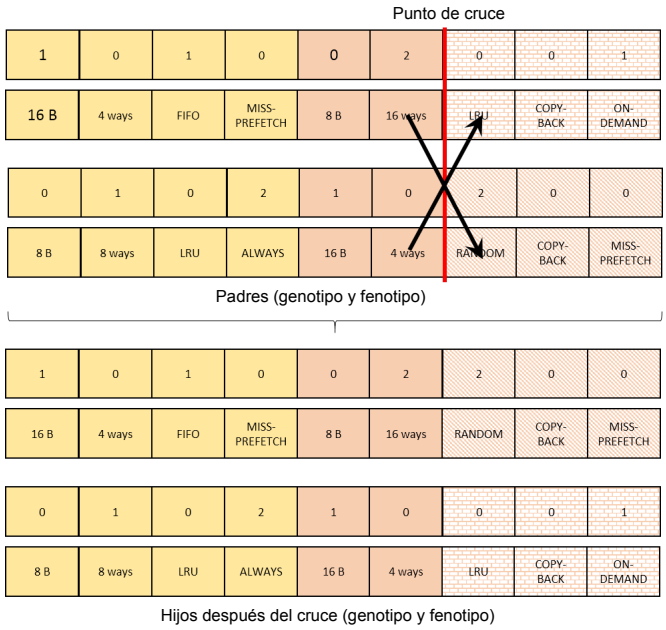


Figura 5.10: Único punto de cruce.

Para NSGA-II, se ha utilizado un único punto de cruce y el operador de mutación clásico (integer-flip) con las probabilidades recomendadas en [150]. El operador de cruce con un único punto de cruce se muestra en la Figura 5.10, donde se selecciona un punto aleatorio en el cromosoma que se usa para generar dos hijos. De forma similar, el operador de mutación clásico (*integer-flip*), se representa en la Figura 5.11. Se genera aleatoriamente un entero para todos los genes que se deben cambiar (según la probabilidad de mutación) y siempre restringidos a los límites del gen correspondiente. Siguiendo el ejemplo dado en la Figura 5.11, se cambia

Figura 5.11: Operador de mutación clásico (Integer-flip).

el tercer gen, modificando su valor de "1" a "0", que en el fenotipo se traduce en cambiar del algoritmo de reemplazo FIFO a LRU, respectivamente.

5.5.4. Experimentos

El entorno de simulación propuesto consiste en un procesador Intel(R) Core(TM) i7-3770 CPU @ 3.40GHz con 16 GB de memoria RAM, con un sistema operativo GNU/Linux Debian 7 corriendo en una versión paralela de maestro-esclavo[2] de NSGA-II, con 8 esclavos. Los resultados experimentales se basan en la arquitectura ARM ampliamente extendida en dispositivos empotrados multimedia. ARM920T es un procesador empotrado típico ampliamente utilizado en tabletas, teléfonos inteligentes y decodificadores, como el teléfono móvil Motorola Q9m Verizon, el navegador GPS HS-3502, etc. El procesador ARM920T pertenece a la familia de procesadores de propósito general ARM9TDMI, que tiene un único núcleo de procesador sobre un dispositivo con arquitectura Harvard. Por defecto, el procesador ARM920T implementa cache de instrucciones y de datos separadas de 16 KB.

Configuración de los experimentos

Para evaluar la efectividad del enfoque propuesto, se ha seleccionado un subconjunto del conjunto de aplicaciones Mediabench, como aplicaciones objetivo.

[2]Un procesador maestro distribuye la evaluación entre los procesadores esclavos que una vez que finalizan envían los resultados al procesador maestro. Este paradigma permite reducir de forma considerable el tiempo de ejecución.

Tabla 5.6: Parámetros del algoritmo NSGA-II.

Número de generaciones	250
Tamaño de la población	100
Longitud del cromosoma	9
Probabilidad de cruce	$0,9$
Probabilidad de mutación	$1/9$

Aunque la metodología propuesta se puede utilizar con cualquier tipo de aplicación, se ha seleccionado el conjunto de aplicaciones Mediabench por la diversidad en cuanto a tamaño de bloque, que proporciona heterogeneidad al espacio de exploración. Usando esta metodología, se ha diseñado una memoria cache óptima de primer nivel con un tamaño fijo para instrucciones y datos, similar a la que disponen algunos dispositivos que tienen un procesador ARM920T. Posteriormente, con objeto de validar el marco de optimización se han incorporado dos plataformas hardware adicionales. Se han simulado doce aplicaciones Mediabench: *cjpeg*, *djpeg*, *mpegdec*, *mpegenc*, *gsmdec*, *gsmenc*, *epic*, *unepic*, *pegwitdec*, *pegwitenc*, *rawcaudio* y *rawdaudio*, todas ellas con sus entradas estándar. Como se ha mencionado anteriormente, sus trazas se han generado mediante la herramienta Trimaran. Para el caso particular de la arquitectura ARM, Trimaran trabaja con SimpleScalar. Ambas herramientas se han modificado para obtener las trazas de aplicación conforme a los requerimientos del simulador de cache Dinero IV, que es llamado continuamente desde la implementación paralela de NSGA-II para evaluar cada solución candidata.

Cada aplicación ha sido simulada para $7,5 \times 10^7$ instrucciones con el fin de alcanzar un equilibrio entre el tiempo de simulación, tamaño de las trazas de programa generadas y un número apropiado de instrucciones. Además, para cada aplicación, NSGA-II se ha ejecutado 30 veces.

La Tabla 5.6 muestra la configuración NSGA-II. Se ha usado un único punto de cruce y el operador de mutación clásico (integer-flip), como se ha mencionado anteriormente. El número de generaciones e individuos han sido establecidos tras realizar varias pruebas preliminares.

Resultados de la optimización

A continuación, se muestran y analizan todos los resultados obtenidos en este marco de optimización propuesto. Las figuras 5.12 y 5.13 muestran los frentes de Pareto obtenidos con el marco de optimización desarrollado. Cada punto en las gráficas representa una configuración de cache y el correspondiente tiempo de ejecución y energía, obtenido mediante las ecuaciones (5.1) and (5.3), respectivamente.

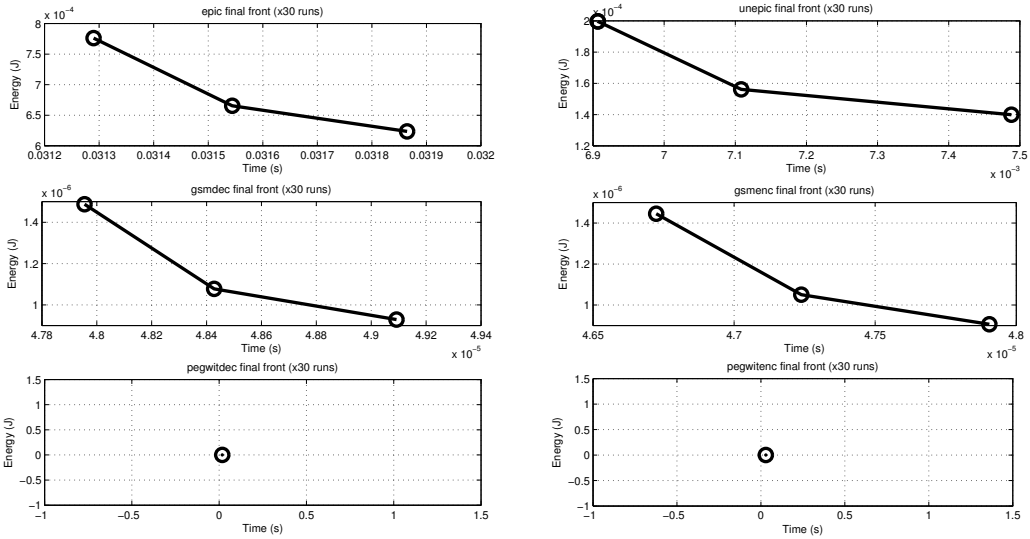


Figura 5.12: Representación del frente de Pareto para *epic*, *unepic*, *gsmdec*, *gsmenc*, *pegwitdec* y *pegwitenc*. Aquellas gráficas donde aparece un único punto que parece representar a única solución, y no un frente de Pareto. Sin embargo, estos puntos representan a varias configuraciones de cache que obtienen los mismos valores para las dos funciones objetivo. En el caso de pegwitdec y pegwitenc son dos las configuraciones de cache que alcanzan los mismos valores objetivo.

En las figuras 5.12 y 5.13, se puede observar que el algoritmo obtiene como mínimo una configuración de cache optimizada para cada aplicación. De hecho, se confirma la hipótesis de que tanto tiempo de ejecución como energía son objetivos contradictorios. Es interesante señalar que en algunas gráficas se puede observar un único punto, que parece representar una única solución, en lugar de un frente de Pareto. Sin embargo, estos puntos individuales representan más de una configuración de cache que alcanzan los mismos valores en el espacio objetivo. Para aclarar este punto, en la Tabla 5.7, se muestra el conjunto de Pareto obtenido para cada aplicación, junto con sus respectivos valores objetivo. La aplicación *mpegdec*, por

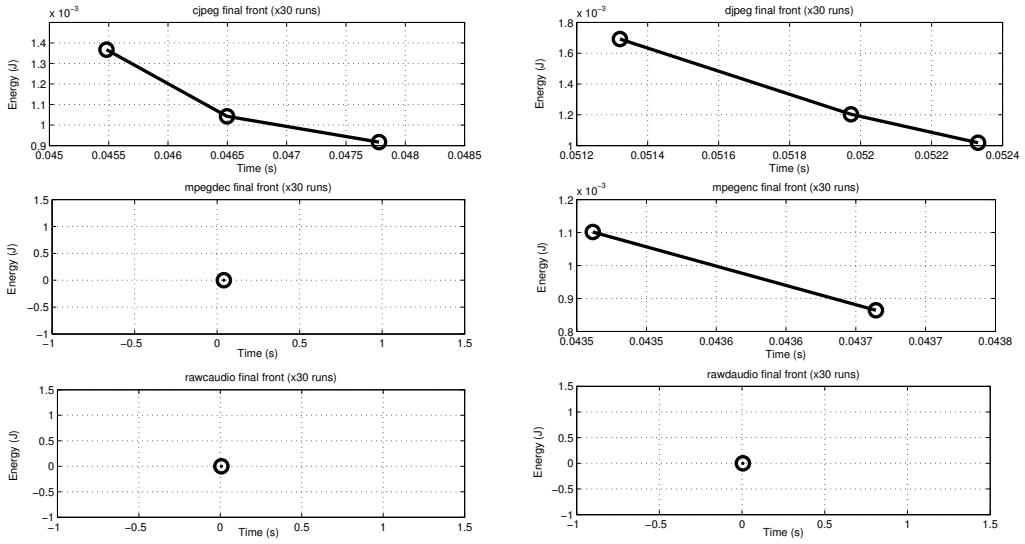


Figura 5.13: Representación del frente de Pareto para *cjpeg*, *djpeg*, *mpegdec*, *mpegenc*, *rawcaudio* y *rawdaudio*. Aquellas gráficas donde aparece un único punto que parece representar a única solución, y no un frente de Pareto. Sin embargo, estos puntos representan a varias configuraciones de cache que obtienen los mismos valores para las dos funciones objetivo. En el caso, por ejemplo, de *rawcaudio* y *rawdaudio* son 17 y 8 las configuraciones de cache que alcanzan los mismos valores objetivo, respectivamente.

Cache de Instrucciones				Cache de Datos				
8 B	4 vías	LRU	ALWAYS	8 B	4 vías	LRU	COPY-BACK	MISS-PREFETCH
8 B	4 vías	LRU	ALWAYS	8 B	4 vías	LRU	WRITE-THROUGH	MISS-PREFETCH

Figura 5.14: Configuraciones de cache compartidas por todos los conjuntos de Pareto, en 9 de las 12 aplicaciones. Las tres aplicaciones restantes, se han evaluado con estas dos configuraciones y los porcentajes de mejora alcanzados son similares a los obtenidos con su mejor configuración.

ejemplo, muestra 2 configuraciones de cache diferentes para los mismos valores objetivo. Lo mismo sucede con *rawcaudio*, con 17 configuraciones distintas con los mismos valores objetivo. Este es un excelente resultado dado que permite simplificar el proceso de selección de una buena configuración de cache, para todas las aplicaciones abordadas.

En este sentido, la Tabla 5.7 muestra que hay dos configuraciones de cache que coinciden en nueve de las 12 aplicaciones utilizadas en este estudio. Estas dos configuraciones se muestran en la Figura 5.14.

Mpegdec, *pegwitdec* y *pegwitenc* son las únicas aplicaciones que no comparten

Tabla 5.7: Conjunto de Pareto para un tamaño de cache de 16 KB.

Aplicación	LI	WI	RI	SI	LD	WD	RD	AD	SD	Tiempo Ej,	Energía
mpegdec	8	4	RANDOM	always	8	4	LRU	Write-back	Miss-prefetch	0,04116	0,00082
								Write-through		0,04116	0,00082
mpegenc	8	4	LRU	always	8	4	LRU	Write-back	Miss-prefetch	0,04371	0,000864
								Write-through		0,04371	0,000864
					32			Write-back		0,04351	0,001101
								Write-through		0,04351	0,001101
cjpeg	8	4	LRU	always	8	4	LRU	Write-back	Miss-prefetch	0,04778	0,000917
								Write-through		0,04778	0,000917
					16			Write-back		0,04649	0,001043
								Write-through		0,04649	0,001043
					32			Write-back		0,04548	0,001367
								Write-through		0,04548	0,001367
djpeg	8	4	LRU	always	8	4	LRU	Write-back	Miss-prefetch	0,05233	0,00102
								Write-through		0,05233	0,00102
					16			Write-back		0,05197	0,00120
								Write-through		0,05197	0,00120
					32			Write-back		0,05132	0,00169
								Write-through		0,05132	0,00169
epic	8	4	LRU	always	8	4	LRU	Write-back	Miss-prefetch	0,03187	0,00062
								Write-through		0,03187	0,00062
					16			Write-back		0,03154	0,00067
								Write-through		0,03154	0,00067
					32			Write-back		0,03129	0,00078
								Write-through		0,03129	0,00078
unepic	8	4	LRU	always	8	4	LRU	Write-back	Miss-prefetch	0,007488	0,0001399
								Write-through		0,007488	0,0001399
					16		RANDOM	Write-back		0,007108	0,0001561
								Write-through		0,007108	0,0001561
					32			Write-back		0,0069067	0,0001997
								Write-through		0,0069067	0,0001997
			RANDOM					Write-back		0,0069068	0,0001997
								Write-through		0,0069068	0,0001997
rawcaudio	8	4	FIFO	always	8	4	FIFO	Write-back	Miss-prefetch	0,00673	0,00013
								Write-through		0,00673	0,00013
							LRU	Write-back		0,00673	0,00013
								Write-through		0,00673	0,00013
							RANDOM	Write-back		0,00673	0,00013
								Write-through		0,00673	0,00013
			LRU				FIFO	Write-back		0,00673	0,00013
								Write-through		0,00673	0,00013
							LRU	Write-back		0,00673	0,00013
								Write-through		0,00673	0,00013
							RANDOM	Write-back		0,00673	0,00013
								Write-through		0,00673	0,00013
			RANDOM				FIFO	Write-back		0,00673	0,00013
								Write-through		0,00673	0,00013
							LRU	Write-back		0,00673	0,00013
								Write-through		0,00673	0,00013
rawdaudio	8	4	LRU	always	8	4	FIFO	Write-back	Miss-prefetch	0,00492	0,000098
								Write-through		0,00492	0,000098
							LRU	Write-back		0,00492	0,000098
								Write-through		0,00492	0,000098
							RANDOM	Write-back		0,00492	0,000098
								Write-through		0,00492	0,000098
			RANDOM					Write-back		0,00492	0,000098
								Write-through		0,00492	0,000098
gsmdec	8	4	LRU	always	8	4	LRU	Write-back	miss-prefetch	4,91E-005	9,29E-007
								Write-through		4,91E-005	9,29E-007
					16			Write-back		4,84E-005	1,08E-006
								Write-through		4,84E-005	1,08E-006
					32			Write-back		4,80E-005	1,49E-006
								Write-through		4,80E-005	1,49E-006
gsmenc	8	4	LRU	always	8	4	LRU	Write-back	miss-prefetch	0,000048	0,0000009
								Write-through		0,000048	0,0000009
					16			Write-back		0,000047	0,0000011
								Write-through		0,000047	0,0000011
					32			Write-back		0,000047	0,0000014
								Write-through		0,000047	0,0000014
pegwitdec	8	4	RANDOM	always	8	4	LRU	Write-back	on-demand	0,01795	0,00034
								Write-through		0,01795	0,00034
pegwitenc	8	4	LRU	always	8	4	LRU	Write-back	on-demand	0,02952	0,00055
								Write-through		0,02952	0,00055

esta configuración de cache. Las mejores configuraciones encontradas producen un ahorro del 63, 45 % y 91, 68 % para *mpegdec*, 60, 09 % y 92, 43 % para *pegwitdec* y 59, 74 % y 92, 55 % para *pegwitenc* en tiempo de ejecución y consumo de energía, respectivamente. Sin embargo, se ha detectado que utilizando la configuración de cache de la Figura 5.14, se ahorra aproximadamente la misma cantidad en tiempo de ejecución y energía (61, 39 % y 90, 98 % para *mpegdec*, 60, 05 % y 91, 83 % para *pegwitdec* y 58, 89 % y 92, 14 % para *pegwitenc*). Ello nos permite afirmar que, definitivamente, estos son realmente buenos resultados para unificar el proceso de selección de una configuración de cache optimizada, para un conjunto de aplicaciones objetivo.

Sin embargo, el número de puntos en el frente de Pareto es pequeño comparado con el tamaño del espacio de búsqueda, 64000 alternativas aproximadamente, como se calculó en la sección 5.5.1. Esto se podría producir porque NSGA-II ha encontrado un óptimo global o el algoritmo cae en un óptimo local muy sólido. Para aclarar este punto, se ha calculado el indicador de hipervolumen (I_H^-) para cada ejecución individual. Esta métrica calcula el volumen, en el espacio objetivo, cubierto por los miembros de un conjunto de soluciones no dominadas Q [143]. Siendo v_i el volumen contenido por la solución $i \in Q$. Entonces, se encuentra la unión de todos los hipercubos y se calcula su hipervolumen (I_H) como:

$$I_H(Q) = \bigcup_{1}^{|Q|} v_i \tag{5.5}$$

Debido a que no se proporciona el conjunto de referencia, éste se toma como $I_H(R) = 0$.

Para mayor claridad, la Figura 5.15 muestra, en un espacio de 2 objetivos, un ejemplo de un frente óptimo de Pareto (curva continua), el conjunto de soluciones obtenidas por un algoritmo de optimización determinado (círculos no sombreados y sombreados en negro), el subconjunto de soluciones no dominadas (círculos sombreados en negro) que forman el frente de Pareto y el subconjunto de soluciones dominadas (círculos no sombreados). Más información sobre este indicador se presenta en el capítulo 3, sección 3.3.2.

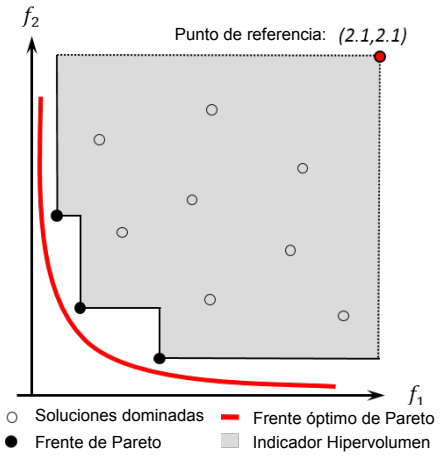


Figura 5.15: Ilustración, en el espacio de dos objetivos, de los conceptos de dominancia, frente óptimo de Pareto, frente de Pareto e indicador hipervolumen. Para este último, se toma el vector objetivo (2.1,2.1) como punto de referencia.

Tabla 5.8: Métrica hipervolumen (S-metric).

Aplicación	**Media**	**STD**
epic	$-0,61$	0
unepic	$-0,69$	0
jcpeg	$-0,61$	0
jdpeg	$-0,47$	0
gsmdec	$-0,64$	0
gsmenc	$-0,62$	$1,18 \times 10^{-16}$
mpegenc	$-0,21$	0

La Tabla 5.8 muestra todos los hipervolúmenes promedio para la mayoría de las 30 diferentes ejecuciones y 7 aplicaciones. No se ha calculado el indicador hipervolumen para todas las aplicaciones porque en algunas de ellas sólo se ha obtenido la misma solución en cada una de las 30 ejecuciones. Para el enfoque aquí propuesto, cuanto menor es el valor del hipervolumen mejor es la población obtenida. Hay que destacar que la desviación estándar es casi 0 para las siete aplicaciones, por tanto, NSGA-II está encontrando el mismo frente de Pareto. Dado que siempre se comienza con una población inicial aleatoria diferente de cromosomas, estos frentes podrían probablemente ser el *Óptimo de Pareto*.

Comparación con una cache base

Para analizar el nivel de mejora usando el enfoque de optimización propuesto con NSGA-II, se comparan los resultados con aquellos obtenidos por una configuración cache base. La configuración base aparece en los dispositivos mencionados anteriormente (el teléfono móvil Motorola Q9m Verizon, el navegador GPS HS-3502, entre otros). Esta cache tiene los siguientes valores de configuración:

- Cache de instrucciones: Tamaño cache: 16 KB; Tamaño bloque: 16; Asociatividad: 64; Algoritmo reemplazo: LRU: Política prebúsqueda: *on-demand*;
- Cache de datos: Tamaño cache: 16 KB; Tamaño bloque: 16; Asociatividad: 64; Algoritmo reemplazo: LRU: Política prebúsqueda: *on-demand*; Política de escritura: *copy-back*;

Se ha calculado el tiempo de ejecución y la energía para una cache base siguiendo el modelo desarrollado en las sección 5.5.2. A continuación, se compara cada punto en los frentes de Pareto reflejados en las figuras 5.12 y 5.13 con las métricas de la configuración base usando las siguientes ecuaciones:

$$\text{Improvement}_{\text{execTime}} = 100 \times \frac{T_{\text{baseline}} - T_{\text{optimized}}}{T_{\text{baseline}}} \tag{5.6}$$

$$\text{Improvement}_{\text{Energy}} = 100 \times \frac{E_{\text{baseline}} - E_{\text{optimized}}}{E_{\text{baseline}}} \tag{5.7}$$

donde T_{baseline} y $T_{\text{optimized}}$ son el tiempo de ejecución de la configuración base y las cache optimizadas, respectivamente. De la misma forma, E_{baseline} y $E_{\text{optimized}}$ son la energía calculada para la configuración base y las cache optimizadas, respectivamente.

En las figuras 5.16 y 5.17 se muestra el porcentaje de mejora que cada uno de los puntos en el frente de Pareto resultado, alcanza para los dos objetivos y para las 12 aplicaciones. Cada punto del frente de Pareto se representa en las barras con un color diferente. La Figura 5.16 describe el porcentaje de mejora en tiempo de ejecución, mientras que la Figura 5.17 representa el nivel de mejora en consumo de energía. Como se puede ver, el enfoque propuesto en esta tesis consigue una mejora

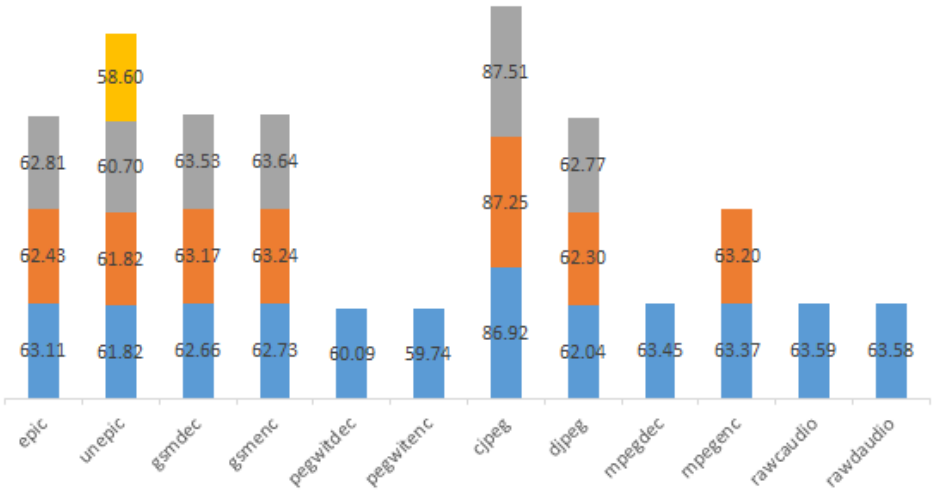


Figura 5.16: Tiempo de ejecución del frente de Pareto respecto a la configuración base (las etiquetas representan el porcentaje de mejora en tiempo de ejecución. Cada color representa un punto en el frente no dominado. Por ejemplo, *unepic* tiene 4 puntos, mientras que *pegwitdec* tiene un único punto.

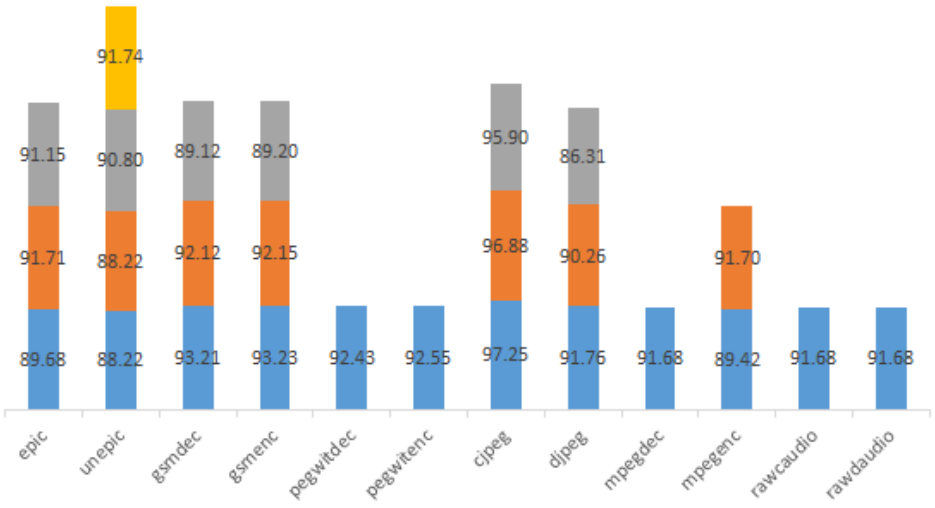


Figura 5.17: Consumo de energía respecto a la configuración base (las etiquetas representan el porcentaje de mejora en consumo de energía). Cada color representa un punto en el frente no dominado, como en la Figura 5.16.

significativa en ambos objetivos. En este sentido, la Tabla 5.9 muestra estas mejoras promedio para el frente de Pareto resultante y sus valores promedios en la última fila. El método de optimización presentado es capaz de llegar a configuraciones de cache que son, en promedio, un $64,43\,\%$ y $91,69\,\%$ mejor en tiempo de ejecución

Tabla 5.9: Porcentaje de mejora, promedio para el frente de Pareto resultante y objetivo individual vs. la configuración base de cache.

Aplicación	Tiempo Ejecución	Consumo Energía
epic	62,79	90,85
unepic	60,74	89,75
gsmdec	63,12	91,48
gsmenc	63,21	91,52
pegwitdec	60,09	92,43
pegwitenc	59,74	92,55
cjpeg	87,23	96,68
djpeg	62,37	89,44
mpegdec	63,45	91,68
mpegenc	63,28	90,56
rawcaudio	63,59	91,68
rawdaudio	63,58	91,68
Promedio	**64,43**	**91,69**

y energía, respectivamente. Para entender mejor este alto porcentaje de mejora, es necesario realizar la comparación de la configuración base, por ejemplo, con la configuración optimizada de la Figura 5.14. En primer lugar, la configuración base tiene un tamaño de bloque de 16 bytes, mientras que la configuración optimizada tiene un tamaño de bloque de 8 bytes. Llevar 16 bytes de memoria principal a memoria cache consume más energía que mover 8 bytes. En segundo lugar, la configuración base tiene una asociatividad de 64 vías frente a las 4 vías de la optimizada. Esto significa que la configuración base tiene un grado de asociatividad mucho mayor y, por tanto, consume mucho más tiempo y energía en encontrar el bloque, dado que cada etiqueta debe ser comparada 64 veces frente a 4. Finalmente, la política de prebúsqueda de la cache de instrucciones de la configuración base es "*on-demand*", mientras que en la optimizada es "*always*". Las instrucciones son normalmente cargadas desde posiciones de memoria consecutivas (descartando las instrucciones de salto), y así, la política de prebúsqueda "*on-demand*" consumirá más tiempo que la política optimizada "*always*" [10].

En lo que se refiere a la convergencia del proceso de optimización, la Tabla 5.10 muestra un resumen de la evolución de los dos objetivos a lo largo del número de generaciones. La columna etiquetada como INI representa el nivel de mejora

alcanzado para cada objetivo promediado sobre la población aleatoria inicial. La columna etiquetada como FINAL representa los mismos valores promediados sobre la población final. La columna AVG muestra las mejoras promedio para cada objetivo y sobre todas las generaciones e individuos, desde INI a END.

Tabla 5.10: Porcentajes de mejora: mejora inicial, promedio y final por aplicación vs. la configuración base.

Aplicación	**Tiempo Ejecución**			**Consumo Energía**		
	INI	**AVG**	**END**	**INI**	**AVG**	**END**
epic	11,40	57,04	63,11	-71,42	77,56	91,71
unepic	12,55	54,97	61,80	-95,27	75,79	91,74
cjpeg	66,13	85,01	87,51	48,03	91,77	97,25
djpeg	12,88	55,26	62,77	-69,11	74,77	91,91
gsmdec	1,25	53,37	63,52	-97,14	77,22	93,19
gsmenc	4,46	55,01	63,65	-71,97	77,58	93,20
rawcaudio	12,63	57,27	63,59	-80,69	76,90	91,68
rawdaudio	10,27	57,50	63,58	-60,05	77,95	91,68
mpegdec	2,18	57,17	63,45	-105,70	76,63	91,68
mpegenc	4,02	57,08	63,37	-66,60	76,97	91,74
pegwitdec	-1,02	51,56	60,09	-146,23	73,00	92,46
pegwitenc	12,07	51,03	59,74	-161,10	71,76	92,55

Como se muestra en la Tabla 5.10, NSGA-II mejora con facilidad el rendimiento de la memoria cache de la línea de base, incluso después de la primera generación. No sucede lo mismo con la energía, donde después de la primera generación, sólo en el caso de *cjpeg*, NSGA-II es capaz de mejorar el consumo de energía de la cache respecto a la configuración base de cache. Afortunadamente, en las primeras generaciones, NSGA-II encuentra rápidamente las configuraciones de cache que mejoran el rendimiento y energía respecto a la configuración base. Sorprendentemente, la mejora del consumo de energía, que comenzó con valores peores, crece rápidamente y alcanza valores mucho mejores que los obtenidos para el tiempo de ejecución (hasta 97 % en el caso de *cjpeg*). En resumen, la Tabla 5.10 demuestra que la metodología de optimización diseñada, incluso comenzando con soluciones iniciales malas, es capaz de alcanzar niveles de mejora altos respecto a una configuración base.

5.5.5. Validación con dos configuraciones cache adicionales

Para validar el marco de optimización se han seleccionado dos nuevas configuraciones de cache base con las que comparar. Concretamente se ha incorporado la memoria cache de dos diferentes plataformas hardware incluidas en algunos dispositivos de Apple de la familia de *SoC Apple AX*, como iPhone 5, iPhone 5S, iPad 2, iPod-touch o iApple-TV. La serie Apple AX integra la familia de procesadores ARM, por ejemplo Cortex-A9 (iPad 2, iPod-touch o iApple-TV) o Cortex-A15 (iPhone 5, iPhone 5S). De acuerdo a esto, las dos nuevas configuraciones de cache son:

- Cache base 2: Tamaño Cache: 32 KB; Tamaño Bloque: 64; Asociatividad: 4; Algoritmo reemplazo: RANDOM: Política prebúsqueda: *always*; Política escritura (Dcache): *copy-back*;
- Cache base 3: Tamaño Cache: 32 KB; Tamaño Bloque: 64; Asociatividad: 2; Algoritmo reemplazo: LRU: Política prebúsqueda: *always*; Política escritura (Dcache): *copy-back*;

Para estas dos configuraciones adicionales de referencia se ha repetido el proceso y se ha calculado el porcentaje de mejora obtenido al comparar cada memoria cache base con la mejor configuración de cache obtenida por el marco de optimización. La Tabla 5.11 muestra los porcentajes obtenidos en tiempo de ejecución y consumo de energía. Es significativo que mientras el ahorro de energía para cada aplicación es similar al de la configuración base 1 (cercano al 90 %), la mejora en tiempo de ejecución se reduce (del 64 % al 24 %, respectivamente). Estas diferencias se deben a la naturaleza de las aplicaciones MediaBench y cada arquitectura base particular. La primera configuración base, un dispositivo GPS para automóviles, está orientada a una aplicación muy específica de navegación, de naturaleza completamente diferente a MediaBench. Ello explica el alto nivel de optimización tanto en tiempo de ejecución como consumo de energía. Por otra parte, las configuraciones base 2 y 3 son dispositivos de propósito general, y sus memorias cache

Tabla 5.11: Porcentaje de mejora para la mejor configuración de cache obtenida frente a las nuevas configuraciones de referencia seleccionadas. Se observa como el ahorro en cuanto a consumo de energía es similar al de la primera configuración base, aproximadamente el 90 % (88, 90 % y 89, 84 %), para ambas configuraciones. Sin embargo, el porcentaje de mejora en tiempo de ejecución se reduce del 64 % al 24 % y 17 % para la configuraciones base 2 y 3, respectivamente. La explicación viene dada por la naturaleza específica de la primera configuración mientras que las 2 últimas son para dispositivos de propósito general, con una gran variedad de tamaños de bloque. El resultado es un nivel de asociatividad más bajo que reduce el tiempo de ejecución y consume más energía.

Aplicación	**Cache Base 2**		**Cache Base 3**	
	Tiempo Ejecución %	**Energía %**	**Tiempo Ejecución %**	**Energía %**
epic	19,80	87,60	11,77	88,66
unepic	9,73	87,28	1,05	88,34
gsmdec	23,97	89,35	16,98	90,27
gsmenc	24,18	89,51	16,98	90,37
pegwitdec	34,26	92,25	28,12	92,72
pegwitenc	35,42	92,61	29,49	93,05
cjpeg	21,11	87,98	13,11	88,98
djpeg	27,38	88,74	21,01	89,93
mpegdec	24,80	88,01	18,61	89,42
mpegenc	22,20	87,65	14,32	88,70
rawcaudio	22,80	87,97	14,68	88,92
rawdaudio	24,13	87,81	16,14	88,77
Promedio	24,15	88,90	16,85	89,84

están orientadas a un amplio rango de tamaños de bloque diferentes. Esto se traduce en una asociatividad baja pero un tamaño grande, o lo que es lo mismo, mejor tiempo de ejecución pero mayor consumo de energía, lo cual explica el bajo nivel de mejora en tiempo de ejecución frente al alto nivel de mejora en energía.

5.5.6. Rendimiento del marco de optimización

Finalmente, con respecto al tiempo de ejecución del proceso de optimización, la arquitectura maestro-esclavo calcula las configuraciones de cache optimizadas en un tiempo medio de 10 horas (0, 42 días) por aplicación. La optimización del conjunto de 12 aplicaciones se ha realizado en 5 días. Teniendo en cuenta que el tiempo medio utilizado por el marco de simulación para evaluar cada configuración de cache individual es igual a 13 segundos, un algoritmo de optimización exhaustivo tardaría casi 10 días en encontrar el frente de Pareto óptimo para una única aplicación, y 117 días para alcanzar el conjunto de los 12 frentes óptimos de Pareto.

Como resultado, un algoritmo NSGA-II maestro-esclavo paralelo obtiene soluciones excelentes (64, 43 % y 91, 69 % mejores en tiempo de ejecución y energía, respectivamente) con una diferencia de más de 3 meses, obteniendo una aceleración de 23, 4 respecto al algoritmo exhaustivo.

Además, para agilizar la ejecución de NSGA-II cada individuo evaluado es almacenado haciendo uso de una tabla *hash*. De esta forma, se evita que si este individuo aparece de nuevo se tenga que volver a evaluar y, por tanto, se reduce el tiempo de ejecución necesario del algoritmo.

5.5.7. Conclusiones

En esta tesis se ha presentado una nueva técnica basada en perfilado estático y optimización multi-objetivo para encontrar la mejor configuración de cache para un sistema empotrado objetivo ampliamente utilizados en dispositivos multimedia, y un conjunto de aplicaciones dado. El proceso se ha dividido en tres fases: la primera de ellas es responsable de realizar la caracterización de memorias cache de acuerdo a los parámetros arquitectónicos básicos. La segunda fase se encarga de obtener las trazas de las aplicaciones objetivo con los patrones de acceso a memoria. Finalmente, la tercera fase aplica el algoritmo evolutivo multi-objetivo, usando NSGA-II y Dinero IV, para evaluar cada aplicación bajo el conjunto de configuraciones candidatas de cache.

El resultado de la optimización es un conjunto de configuraciones de cache que minimiza el tiempo de ejecución y el consumo de energía para cada aplicación, en un tiempo aceptable. Los resultados, por tanto, permiten mejorar el rendimiento e incrementar la vida útil tanto de las baterías como de los dispositivos. Tomando una configuración de cache, comúnmente utilizada en los actuales sistemas multimedia como configuración base, los resultados experimentales muestran una mejora promedio de 64, 43 % y 91, 69 % en tiempo de ejecución y consumo de energía, respectivamente. Adicionalmente, se utilizan dos configuraciones de cache pertenecientes a las familias Cortex A9 y A15, para validar el marco de optimización y se

obtiene un porcentaje de mejora en tiempo de ejecución y consumo de energía del $24,15\,\%$ y $88,90\,\%$ y del $16,85\,\%$ y $89,84\,\%$, respectivamente.

Esta metodología aún necesita de la toma de decisión humana para seleccionar la memoria cache definitiva, la mejor posible para el conjunto de aplicaciones completo. Aunque se ha comprobado que esta no es una tarea demasiado complicada, sería interesante abordar una metodología que permita la optimización automática de todas las aplicaciones objetivo al mismo tiempo. En la siguiente sección, se aborda este proceso mediante el uso de evolución diferencial.

5.6. Optimización del rendimiento de la memoria cache L1 para sistemas empotrados dirigida por Evolución Diferencial

5.6.1. Introducción

El enfoque propuesto en esta sección tiene como referencia el marco teórico explicado en la sección 5.1, tras lo cual se realiza la revisión de antecedentes en la sección 5.2 que son comunes a los algoritmos propuestos en el marco de optimización que se presenta en esta tesis. En las dos secciones anteriores se ha explicado el marco de optimización dirigido por GE y el algoritmo multi-objetivo NSGA-II. Ambos han abordado el proceso de optimización para cada aplicación individual. Sin embargo, tal y como se ha mencionado previamente, la mejor configuración de cache para una aplicación individual podría no ser la mejor para un conjunto completo de aplicaciones y, en consecuencia, incrementaría el consumo de energía. De esta forma, encontrar la mejor configuración de cache para un conjunto de aplicaciones definido comienza con la identificación de aquellas configuraciones de cache que maximizan el rendimiento y minimizan el consumo de energía del conjunto completo.

En este sentido, se ha diseñado un proceso de optimización para identificar la mejor configuración de cache para un conjunto de aplicaciones, que consiste en 4 pasos: (1) simular las aplicaciones para recoger sus patrones de acceso a memoria; (2) explorar el espacio de diseño de las configuraciones de cache, de acuerdo al conjunto de parámetros y sus posibles valores; (3) calcular el rendimiento total y el consumo de energía de las aplicaciones para cada configuración de cache candidata; (4) decidir cual de las configuraciones de cache resultantes se adapta mejor a todas las aplicaciones objetivo. En la sección 5.3, se ha realizado una explicación más detallada de los pasos 1 y 2.

Con este objetivo, se presenta el marco de optimización dirigido por la técnica evolutiva Differencial Evolution (DE) [106], capaz de identificar la mejor configuración de cache que satisface los requerimientos de un conjunto completo de

aplicaciones multimedia dado. El objetivo es encontrar la configuración óptima de cache, que minimiza el tiempo de ejecución y el consumo de energía de la cache, para un conjunto de aplicaciones objetivo. Los resultados experimentales, usando un subconjunto de aplicaciones del conjunto de aplicaciones Mediabench, demuestran que esta propuesta es capaz de encontrar configuraciones de cache que obtienen una mejora superior al 62 % comparadas con una configuración realista de cache, de referencia.

Tiempo de ejecución y consumo de energía se calculan y utilizan para medir el valor objetivo de cada configuración de cache, en cada etapa del proceso de evolución. Los resultados experimentales muestran que DE obtiene resultados excelentes en un tiempo de ejecución reducido (menos de 14 horas en un sistema paralelo maestro-esclavo con 8 esclavos). La fase experimental se lleva a cabo para un conjunto de aplicaciones Mediabench, principalmente para procesamiento multimedia. Como arquitectura base, se ha seleccionado la familia de procesadores ARM9, ampliamente utilizado en dispositivos móviles multimedia, por el diseño flexible de cache que posee.

5.6.2. Metodología de optimización

La metodología del marco de optimización se describe con más detalle en la sección 5.3 y la Figura 5.2 muestra el esquema de trabajo del proceso de optimización. En esta sección se describe la metodología específica del enfoque propuesto con DE. En este enfoque, se considera la mejor configuración de cache, aquella que proporciona el mejor compromiso entre minimizar el tiempo de ejecución y el consumo de energía para un conjunto completo de aplicaciones objetivo.

El proceso de optimización se divide en tres fases, la primera fase se encarga de obtener la caracterización de cache; la segunda obtiene las trazas de programa y la tercera fase es responsable del algoritmo DE, basado en el algoritmo propuesto por Storn y Price en [106], como una técnica adecuada para un amplio rango de problemas de optimización estocásticos continuos. Una vez que se obtienen las ca-

racterizaciones de cache y las trazas de programa, comienza la tercera fase con la aplicación del algoritmo DE. Tras la ejecución, DE devuelve la configuración de cache que obtiene el mejor compromiso entre rendimiento y consumo de energía para todas las aplicaciones objetivo.

Para evaluar cada configuración de cache se utiliza el modelo de rendimiento y energía explicado en la sección 5.3.1, donde también se encuentran recogidas las características físicas de la memoria DRAM.

Fases 1 y 2: Caracterización de cache y perfilado de aplicaciones

Las fases 1 y 2, correspondientes a la caracterización de cache y el perfilado de las aplicaciones, sigue la misma metodología que se ha detallado en la sección 5.3.4.

Fase 3: Proceso de optimización

Después de completar la fase *off-line*, las trazas de programa y la caracterización de cache están disponibles y el proceso de optimización dirigido por el algoritmo DE comienza. En esta sección se describe brevemente esta técnica heurística que ya ha sido abordada en el Capítulo 3 en la sección 3.2.6.

El algoritmo de evolución se centra en el operador de mutación, seguido por un operador de recombinación. DE es fuertemente dependiente de sus parámetros, como el tamaño de la población, el factor de recombinación y la estrategia de mutación seleccionada. Por otra parte, presenta un inherente facilidad para ser paralelizado. Desde el enfoque original, han ido apareciendo diferentes versiones, por ejemplo, Das et al. [136] presentaron un trabajo muy extenso acerca de los cambios realizados sobre la propuesta original.

DE trabaja sobre una población compuesta por N individuos que representan soluciones candidatas al problema dado. Cada solución candidata se define como un vector de d dimensiones identificado por x_i^g, donde i representa la posición dentro de la población y g es la generación actual. Cada vector contiene las variables que definen el problema, cuya longitud está determinada por $x_{i,j}^g$, donde j identifica la variable en el individuo. El algoritmo evoluciona a través de g_M generaciones.

En esta tesis, se utiliza el algoritmo DE (DE/rand). Los valores seleccionados en los experimentos para N, g_M, F y C_r se pueden consultar en la Tabla 5.13. Estos se han establecido siguiendo las recomendaciones de [186] y después de realizar experimentos previos con otros valores.

Por tanto, el enfoque DE explora el espacio de búsqueda de las configuraciones de cache delimitadas por los valores mínimo y máximo de los parámetros. Para cada individuo, correspondiente a cada configuración de cache, se evalúan todas las aplicaciones. Para cada aplicación, se llama al simulador de cache para obtener el número de aciertos y fallos. En este punto, se aplican los modelos de rendimiento y de energía descritos en la sección 5.3.1, y se obtienen el tiempo y la energía por cada acceso a cache. Estos son utilizados para calcular el tiempo de ejecución y el consumo de energía de la configuración de cache actual y todas las aplicaciones objetivo. Los individuos evaluados se almacenan en una tabla *hash* que permite recuperarlos de forma rápida, en caso de volver a aparecer y, de esta forma, se reduce de forma significativa el tiempo necesario para la ejecución del algoritmo. Después de completar el número de generaciones parametrizadas, DE obtiene una configuración de cache optimizada para el conjunto de aplicaciones objetivo.

A tal efecto, se ha diseñado una función objetivo, que consiste en la suma ponderada del tiempo de ejecución y consumo de energía de las aplicaciones objetivo. Se le ha asignado el mismo peso a ambas características, seleccionando una configuración de cache como configuración base para normalizar tiempo de ejecución y consumo de energía. Por tanto, la función objetivo definida en (5.8) dirige el proceso de evaluación para una configuración concreta de cache c_i.

$$f(c_i) = 0{,}5 \times \frac{T(c_i)}{T(Baseline)} + 0{,}5 \times \frac{E(c_i)}{E(Baseline)} \tag{5.8}$$

En este sentido, $T(c_i)$, $E(c_i)$ son la suma de los tiempos de ejecución y la suma de los consumos de energía del conjunto completo de aplicaciones obtenidas bajo la configuración actual de cache, c_i. Es decir,

$$T(c_i) = \sum_{j=1}^{N} t_j(c_i) \quad \text{and} \quad E(c_i) = \sum_{j=1}^{N} e_j(c_i)$$

, donde N es el número de aplicaciones objetivo. Asimismo, $T(Baseline)$ y $E(Baseline)$ corresponden a la suma del tiempo de ejecución y consumo de energía de las aplicaciones objetivo bajo la configuración base de cache. La normalización permite escalar y operar correctamente con las unidades de diferentes métricas como son el tiempo y la energía. El objetivo es encontrar el mejor compromiso entre la mejora del rendimiento y el consumo de energía, así cada término de la ecuación se pondera al 50 %, como se muestra en la Ecuación (5.8). La mejor configuración de cache en DE será aquella que minimice el valor objetivo.

Finalmente, la optimización de la cache de instrucciones y datos consiste en ajustar los parámetros que conjuntamente con sus valores se representan en la Figura 5.1 de la sección 5.3.3, donde se explica y cuantifica de forma pormenorizada el espacio de búsqueda. Este enfoque con DE utiliza el espacio de diseño completo representado en la Figura 5.1. Por tanto, la combinación de todos los valores diferentes componen el espacio de búsqueda, donde una configuración de cache consiste en 11 parámetros con los valores permitidos que se muestran en ella.

Así, de acuerdo a la Figura 5.1, un individuo representado como se muestra en la Figura 5.18 (superior), será decodificado como la configuración de cache que se muestra en la Figura 5.18 (inferior).

Cache Instrucciones					Cache Datos					
1	0	0	0	0	2	1	2	1	2	0

Cache Instrucciones					Cache Datos					
1 KB	8 B	1 ways	LRU	MISS-PREFETCH	2KB	16B	4 ways	FIFO	ALWAYS	COPY-BACK

Figura 5.18: Configuración de cache. La tabla superior es el vector de enteros (genotipo), codificado por el algoritmo DE. La tabla inferior representa los parámetros de la cache actual una vez que se ha decodificado el genotipo (fenotipo).

En resumen, este problema define su espacio de búsqueda como un conjunto de configuraciones de cache. Sin embargo, DE representa el genotipo de sus individuos como vectores de parámetros numéricos, lo que hace necesario implementar un proceso de mapeo. Así, el cromosoma de cada individuo consiste en un vec-

Tabla 5.12: Aplicaciones Mediabench.

Aplicación	Descripción
JPEG	Método C estándar para comprimir-descomprimir imágenes.
MPEG	Codificador-decodificador MPEG2 para transmisión digital en alta calidad.
GSM	Algoritmo de reconocimiento de voz basado en el estándar europeo GSM 06.10.
PEGWIT	Autenticación y encriptación de clave pública.
EPIC	Compresión y descompresión de datos de imágenes.
ADPCM	Algoritmos de compresión y descompresión de voz.

tor de valores enteros, correspondiente a valores asignados a cada opción de los parámetros. Una vez generado, se evalúa cada configuración de cache en todas las aplicaciones abordadas y se obtiene el valor objetivo, de acuerdo a la función de coste descrita en la Ecuación (5.8). Una vez que el número de generaciones se ha completado, DE suministra la configuración de cache que minimiza el valor de la función objetivo de las aplicaciones seleccionadas.

5.6.3. Resultados experimentales

Para evaluar el enfoque heurístico propuesto se ha seleccionado un subconjunto de las aplicaciones Mediabench. Este conjunto de aplicaciones reúne las características representativas de las aplicaciones multimedia soportadas por los dispositivos móviles actuales, como restricciones de tiempo, cómputo intensivo, alta disponibilidad de recursos, por ejemplo memoria, etc. En la Tabla 5.12 se muestra un breve resumen del conjunto de aplicaciones seleccionado.

Las aplicaciones se evalúan bajo cada configuración de cache para analizar su comportamiento. De esta forma, los experimentos se han llevado a cabo en un sistema con procesador Intel(R) Core(TM) i7-3770 CPU @ 3.40GHz con 16 GB de memoria RAM, con sistema operativo GNU/Linux Debian 7, usando un algoritmo DE paralelo maestro-esclavo con 8 esclavos. De esta forma, es posible realizar de forma simultánea la evaluación de ocho posibles configuraciones de cache y así, reducir de forma considerable el tiempo necesario para el proceso de evaluación.

Tabla 5.13: Parámetros para DE.

Parámetro	Valor
Número de ejecuciones	30
Número de generaciones (g_M)	100
Tamaño de población (N)	25
Factor de recombinación (C_r)	0, 3
Factor de mutación (F)	0, 5

Configuración de los experimentos

El marco experimental se ha diseñado para la arquitectura on-chip ARM, considerando un primer nivel de cache, con memorias cache separadas de instrucciones y datos, conectadas a la memoria principal. De esta forma, se aborda el proceso de optimización utilizando la metodología explicada en la sección 5.6.2, las seis aplicaciones anteriormente citadas y DE como algoritmo evolutivo.

El marco experimental seleccionado, así como todos los parámetros implicados en el espacio de búsqueda están ampliamente descritos en la sección 5.3.3 y mostrados en la Figura 5.1. En la sección 5.6.2 se ha puntualizado el alcance de los experimentos en cuanto al número de aplicaciones utilizadas que, en este enfoque, son 6 aplicaciones, una por cada aplicación Mediabench citada. Siguiendo el mismo criterio que los enfoques con GE y NSGA-II, cada aplicación se ha simulado un máximo de $7{,}5 \times 10^7$ instrucciones, o la aplicación completa si el número total de instrucciones no alcanza este valor establecido. El objetivo es alcanzar un equilibrio entre el tiempo de simulación, tamaño de las trazas de programa y un número apropiado de instrucciones a ejecutar.

Los problemas de optimización normalmente pueden caer en un óptimo local y para reducir la probabilidad de que esto ocurra, el algoritmo se ha ejecutado 30 veces. La Tabla 5.13 muestra la parámetros del algoritmo DE, los cuales han sido seleccionados después de realizar experimentos previos con valores como 0,5 y 0,9 para el factor de recombinación, y 1 para el factor de mutación.

Como se establece en la Ecuación (5.8), el algoritmo necesita de una configu-

ración base de cache para calcular el valor de aptitud de un individuo. Para estos experimentos, se ha selecciona una configuración base de cache que es similar a la cache del primer núcleo de la consola portátil de juegos GP2X, la cual dispone de un procesador ARM940T dual a 200 MHz. El tiempo de ejecución y el consumo de energía para la cache base tiene que ser calculado siguiendo los modelos descritos en la sección 5.3.1. Los parámetros de la configuración base se muestran en la Tabla 5.14.

Tabla 5.14: Configuración base de cache

Cache de Instrucciones					
Tamaño	**Tam. Bloque**	**Asoc.**	**Alg. Reempl.**	**Alg. Búsqueda**	
16 KB	32 B	4	LRU	On-Demand	
Cache de Datos					
Tamaño	**Tam. Bloque**	**Asoc.**	**Alg. Reempl.**	**Alg. Búsqueda**	**Pol. Escritura**
16 KB	32 B	4	LRU	On-demand	Copy-Back

Resultados de la optimización con DE

Para cada instancia, el algoritmo DE se ha ejecutado 30 veces con la configuración mostrada en la Tabla 5.13. Para cada ejecución, DE devuelve el mejor individuo encontrado después del proceso de evolución, en términos de valor objetivo. Todas las aplicaciones objetivo se evalúan para cada individuo, por tanto el mejor individuo representa una configuración de cache que mejora el consumo de energía y el rendimiento para el conjunto de aplicaciones completo. La Figura 5.19 representa el resultado de las 30 ejecuciones evaluadas de manera individual respecto a los valores objetivo en porcentaje de mejora mínimo (amarillo), promedio (rojo) y máximo (azul). El valor objetivo se expresa como un porcentaje de mejora, que se calcula mediante la asignación del mismo peso (50 %) para ambas métricas, tiempo de ejecución y consumo de energía con respecto a la configuración base, tal y como aparece en la Ecuación (5.8).

Una vez que el algoritmo DE se ha ejecutado, cada ejecución devuelve el menor valor objetivo, es decir aquel que ofrece el porcentaje de mejora máximo, como

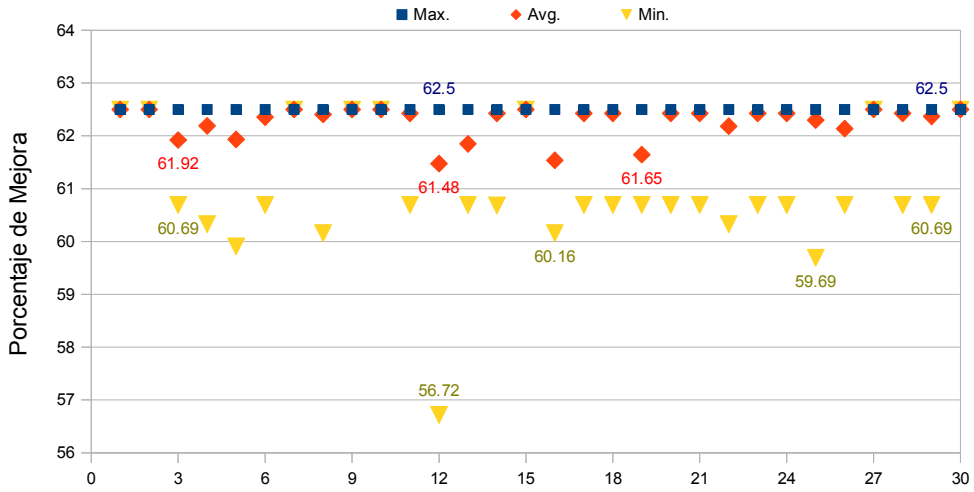


Figura 5.19: Porcentaje mínimo, máximo y promedio durante 100 generaciones y 30 ejecuciones obtenidas mediante DE para un conjunto de aplicaciones. Los valores representados en amarillo, rojo y azul indican los porcentajes de mejora mínimo, promedio y máximo, respectivamente, alcanzados durante las 100 generaciones de cada ejecución. En esta gráfica, no se tienen en cuenta aquellas configuraciones que no mejoran el valor de aptitud con respecto a la cache base.

se observa mediante los cuadrados azules en la Figura 5.19. En esta gráfica, las configuraciones de cache que no mejoran el valor objetivo de la configuración base no se tienen en cuenta.

Es interesante destacar que los porcentajes de mejora son alcanzados no solo por una configuración individual sino por un conjunto de configuraciones de cache. De hecho, la mayoría de las buenas soluciones obtenidas por el algoritmo DE aplicado representa a varias configuraciones de cache. En resumen, el algoritmo suministra un conjunto de configuraciones de cache que mejora el rendimiento y el consumo de energía (en relación con las operaciones de memoria), respecto a la configuración base de cache seleccionada.

Es interesante mencionar que el espacio de diseño completo contiene 10616832 configuraciones de cache posibles. DE puede generar hasta 75750 posibles individuos de acuerdo al conjunto de los parámetros DE (100 generaciones x 25 individuos x 30 ejecuciones, además de la población inicial para cada ejecución), sin embargo, se ha calculado que el 70,34 % representan configuraciones viables de cache. A pesar de que las soluciones generadas se pueden repetir, el algoritmo DE

ha evaluado 19352 configuraciones diferentes de cache en las pruebas realizadas. Además, el tiempo necesario para evaluar cada solución depende del simulador de cache y del tamaño de la traza de programa, por tanto este proceso consume mucho tiempo. Teniendo en cuenta que cada configuración de cache es evaluada para las seis aplicaciones objetivo y para cada aplicación el simulador de cache utiliza 13 segundos, en promedio, evaluar las 75750 configuraciones posibles para las seis aplicaciones llevaría más de 68 días con un algoritmo exhaustivo. Sin embargo, la implementación de almacenamiento con tablas *hash* permite evaluar sólo las configuraciones diferentes porque los individuos ya evaluados pueden ser recuperados rápidamente y se obtendrían los resultados tras 18 días aproximadamente. La utilización de un entorno paralelo permite reducir el tiempo de simulación a algo más de 2 días y esto produce un ahorro de tiempo del $92,8\,\%$ respecto a una enfoque exhaustivo del algoritmo.

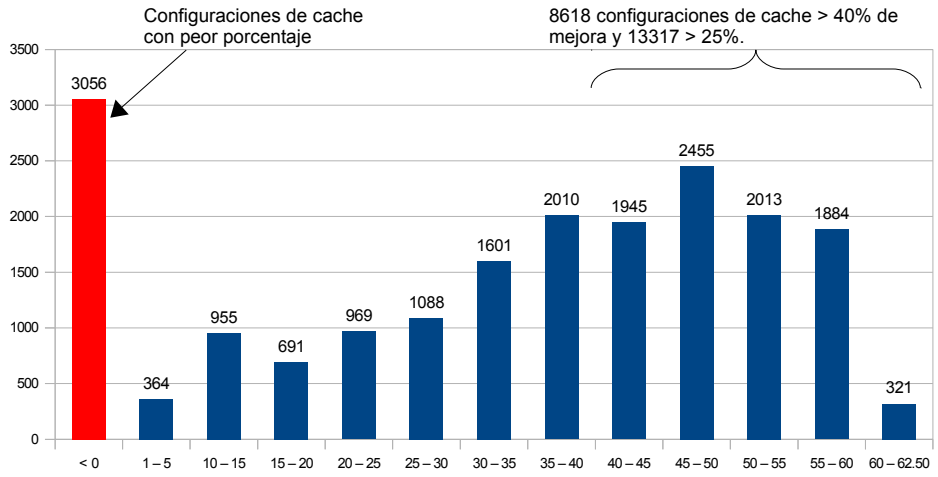


Figura 5.20: Porcentajes de mejora de los valores de aptitud durante 30 ejecuciones del algoritmo DE, para las aplicaciones Mediabench sobre la configuración base. El eje **x** representa los rangos de porcentajes de mejora, mientras que el eje **y** es el número de soluciones diferentes obtenidas para cada intervalo. La barra roja en el gráfico representa las soluciones que obtienen peor valor de aptitud que la configuración base.

La Figura 5.20 presenta un resumen de la información parcial recopilada por el algoritmo DE durante todas las 30 ejecuciones. El eje **x** representa los rangos de porcentajes de mejora obtenidos a través del total de ejecuciones, mientras que el

eje **y** es el número de soluciones diferentes para cada intervalo. El mejor porcentaje de mejora respecto a la configuración base está sobre el 62,5 %. Después de analizar los resultados, este porcentaje lo obtienen 18 configuraciones diferentes. Además, más de 4000 configuraciones de cache alcanzan un porcentaje de mejora superior al 50 %. Sin embargo, evaluar 4000 configuraciones de cache aun conlleva mucho tiempo de evaluación. Después de numerosas pruebas, se determina que para obtener mejoras que rondan el 60 % en todos los casos es suficiente con realizar 10 ejecuciones durante 100 generaciones.

El algoritmo DE ofrece un amplio conjunto de individuos que es capaz de alcanzar mejoras en los dos valores objetivo, tiempo de ejecución y consumo de energía, para el conjunto de aplicaciones objetivo, lo cual representa un resultado excelente. De hecho, todas las soluciones que alcanzan los porcentajes más altos tienen algo en común, los parámetros arquitectónicos de cache, los cuales coinciden para todas las configuraciones tanto para la cache de instrucciones como la cache de datos. A la vista de los resultados, se analizan las 321 soluciones cubiertas por el intervalo identificado como 60 – 62,5 y que representan los porcentajes más altos de mejora.

Tabla 5.15: Conjunto de configuraciones de cache con el porcentaje de mejora más alto.

Cache de Instrucciones					**Cache de Datos**						**% Mejora**
IS	**BS**	**AS**	**Repl.**	**SA**	**DS**	**BS**	**AS**	**Repl.**	**SA**	**Alloc.**	**%**
512	8	1	fifo, lru, random	always	8192	8	2	fifo, lru, random	miss, on-demand	copy-back, write-through	**61.21, 60.67**
512	16	1	fifo, lru, random	always	8192	8	1	fifo	miss, random	copy-back, write-through	**60.04**
1024	8	1	fifo, lru, random	always	8192	8	1	fifo, lru, random	miss, on-demand	copy-back, write-through	**62.50, 61.97**
1024	8	1	fifo, lru, random	always	16384	8	1	fifo, lru, random	miss, on-demand	copy-back, write-through	**61.77, 61.28**
1024	8	1	fifo, lru, random	always	32768	8	1	fifo, lru, random	miss, on-demand	copy-back, write-through	**61.26, 61.05**
1024	8	1	fifo, lru, random	always	8192	16	4	lru	miss, on-demand	copy-back, write-through	**60.69, 60.28**
1024	8	1	fifo, lru, random	always	8192	16	4	fifo	miss	copy-back, write-through	**60.33**
1024	8	1	fifo, lru, random	always	16384	8	2	lru	miss	copy-back, write-through	**60.17**
1024	8	1	fifo, lru, random	always	16384	8	2	fifo	miss	copy-back, write-through	**60.02**
2048	8	1	fifo, lru, random	always	8192	8	1	fifo	miss	copy-back, write-through	**60.32**
2048	8	1	fifo, lru, random	always	16384	8	2	lru	miss	copy-back, write-through	**60.15**
2048	8	1	fifo, lru, random	always	16384	8	1	fifo, lru, random	miss, on-demand	copy-back, write-through	**61.76, 61.28**
2048	8	1	fifo, lru, random	always	32768	8	1	fifo, lru, random	miss, on-demand	copy-back, write-through	**61.25, 61.04**
2048	8	1	fifo, lru, random	always	8192	16	4	lru	miss, on-demand	copy-back, write-through	**60.68, 60.27**

Cada fila de la Tabla 5.15 representa tantas configuraciones de cache como combinaciones diferentes se pueden hacer con todos los valores posibles en cada columna. Por ejemplo, la primera fila contiene 36 configuraciones diferentes de cache ($1 \times 1 \times 1 \times 3 \times 1 \times 1 \times 1 \times 1 \times 3 \times 2 \times 2$). La última columna es el porcentaje de mejora sobre la configuración cache de referencia, en aquellas filas donde apa-

recen dos valores diferentes se debe a que en alguna de las soluciones obtenidas el algoritmo de búsqueda para la cache de datos es *on-demand* (buscar sólo cuando es necesario). Es decir, las diferencias relacionadas a los porcentajes de mejora dentro de la misma fila son provocadas por el algoritmo de búsqueda para la cache de datos, particularmente *on-demand* frente a la mejora obtenida con el algoritmo de búsqueda *miss*. Ante un fallo en la cache, la política de búsqueda *miss* carga el bloque solicitado y el siguiente evitando algunos de los fallos que se producirían por referenciar instrucciones (cache de instrucciones) o datos (cache de datos), cercanos en tiempo o espacio y que necesitarían ser solucionados. Así, cuando el algoritmo *on-demand* no aparece dentro de la misma fila, el porcentaje de mejora es idéntico para cada configuración de cache. En vista de los resultados, las diferencias entre las configuraciones de cache están principalmente relacionadas con los parámetros arquitectónicos típicos (tamaño de cache, tamaño de bloque y asociatividad).

Es interesante resaltar que todas las configuraciones tienen una cache de instrucciones directamente mapeada con un tamaño de cache, principalmente de 8 bytes, con *always* como algoritmo de búsqueda, que se considera como uno de los más eficientes. Respecto a la cache de datos, destaca la presencia de baja asociatividad, donde el tamaño de cache no es demasiado grande para las soluciones con el mejor porcentaje de mejora. Incluso las configuraciones para la cache de datos directamente mapeadas ofrecen mejores resultados para todos los tamaños de cache mostrados en la Tabla 5.15. Además, analizando los resultados, más del 72 % de las configuraciones de cache presentan $asociatividad \leq 4$; el 87 % tienen un tamaño de cache $\leq 16KB$ y alrededor del 83,5 % presenta un tamaño de bloque ≤ 8. Los porcentajes de mejora empeoran cuando los valores de los parámetros arquitectónicos son más altos.

Finalmente, el enfoque propuesto realiza una búsqueda amplia a lo largo del espacio de diseño y es capaz de identificar las mejores configuraciones de cache para un conjunto de aplicaciones objetivo. A pesar de que las aplicaciones seleccionadas pueden presentar patrones de comportamiento diferentes, cada día se desarrollan

nuevas aplicaciones multimedia para sistemas empotrados, las cuales añaden nuevas funcionalidades y requerimientos al sistema. En este marco propuesto se han abordado 6 aplicaciones objetivo, por tanto, otro conjunto de aplicaciones diferente debería pasar por el mismo proceso para alcanzar la configuración óptima.

5.6.4. Conclusiones

En esta tesis, se ha desarrollado un marco de optimización especialmente dirigido al diseño de sistemas empotrados presentes en la mayoría de los dispositivos móviles y, en particular, al subsistema de memoria cache. Se ha considerado el rendimiento y el consumo de energía, ambos relativos a las operaciones de memoria cache. Esta propuesta se basa en perfilado estático y la técnica de computación evolutiva DE, cuyo principal objetivo es explorar y evaluar automáticamente el espacio de diseño de las configuraciones de memoria cache. Finalmente, DE es el responsable de proponer la mejor configuración de cache para el conjunto de aplicaciones objetivo. Los parámetros arquitectónicos clásicos, junto a los algoritmos de reemplazo, búsqueda tanto para la cache de instrucciones como de datos, y la política de escritura para la memoria cache de datos completan los 11 parámetros considerados en esta propuesta.

El marco de optimización se divide en tres fases, las dos primeras fases se encargan de suministrar la caracterización de cache y las trazas de programa; una tercera fase es conducida por el algoritmo DE para obtener la mejor configuración de cache para un conjunto completo de aplicaciones objetivo.

Para determinar los beneficios de esta propuesta se ha incluido un subconjunto de 6 aplicaciones pertenecientes al conjunto Mediabench y los resultados se han comparado con una configuración de cache que actúa de referencia. Los resultados obtenidos muestran que el marco propuesto es capaz de encontrar, al menos, una configuración de cache que mejora más del 62,5 % la configuración de cache base seleccionada. Esto significa una mejora en términos de tiempo de ejecución y consumo de energía. Además, 18 configuraciones de cache comparten este porcen-

taje de mejora y un 22 % de las configuraciones de cache evaluadas alcanzan un porcentaje de mejora superior al 50 %.

A la vista de los resultados, se observa que los parámetros arquitectónicos típicos de cache son los más influyentes. Además, el comportamiento de la cache de datos varía más que el comportamiento de la cache de instrucciones, según los parámetros de cache. Por lo tanto, algunos parámetros de cache, particularmente para la cache de instrucciones, podrían no tenerse en cuenta con el fin de reducir el tiempo necesario del proceso de optimización. También se considera la necesidad de simplificar el marco de optimización para suministrar una forma más eficiente de encontrar la configuración óptima de cache para un conjunto de aplicaciones objetivo.

5.7. Comparativa optimización de cache

Durante el capítulo 5 se ha abordado el problema de la optimización de la memoria cache para sistemas empotrados. Para llevarlo a cabo se ha propuesto un marco de optimización basado en tres técnicas heurísticas diferentes como son GE, NSGA-II y DE. Los resultados obtenidos en cada uno de ellos se han presentado y analizado, de forma independiente, en las secciones 5.4, 5.5, 5.6. En esta sección, se aborda la comparación de los resultados obtenidos entre las tres técnicas metaheurísticas propuestas. Sin embargo, los enfoques abordados mediante GE y NSGA-II llevan a cabo el proceso de optimización para cada aplicación individualmente, mientras que la propuesta realizada con DE lleva a cabo la optimización para una serie de aplicaciones, de forma simultánea. Además, el conjunto de aplicaciones utilizado en el enfoque con DE se ha limitado a 6 con el fin de reducir el tiempo de ejecución del proceso de optimización. De esta forma, la comparación directa de los resultados no es posible.

Por otra parte, los enfoques de GE y NSGA-II utilizan un espacio de búsqueda distinto y la configuración base de cache con la que comparar es distinta, para explorar el espacio de búsqueda en diferentes direcciones. Respecto al espacio de búsqueda, en NSGA-II se trata un único tamaño de cache de $16KB$ y 4 posibles alternativas para el grado de asociatividad, tanto para la cache de instrucciones como de datos. Sin embargo, GE puede seleccionar entre 8 tamaños de cache y 8 grados de asociatividad diferentes, incrementando considerablemente el espacio de búsqueda. En este sentido, para poder comparar los resultados obtenidos con ambas metaheurísticas se ha abordado el proceso de optimización mediante NSGA-II, tomando el mismo espacio de búsqueda diseñado para GE que se puede observar en la Figura 5.21. Es importante destacar, que NSGA-II es multi-objetivo y aborda el proceso de optimización definiendo el tiempo de ejecución y el consumo de energía como los dos objetivos a optimizar, mientras que GE es mono-objetivo y utiliza una función objetivo ponderada. Por tanto, el enfoque con NSGA-II puede llegar a resultados diferentes.

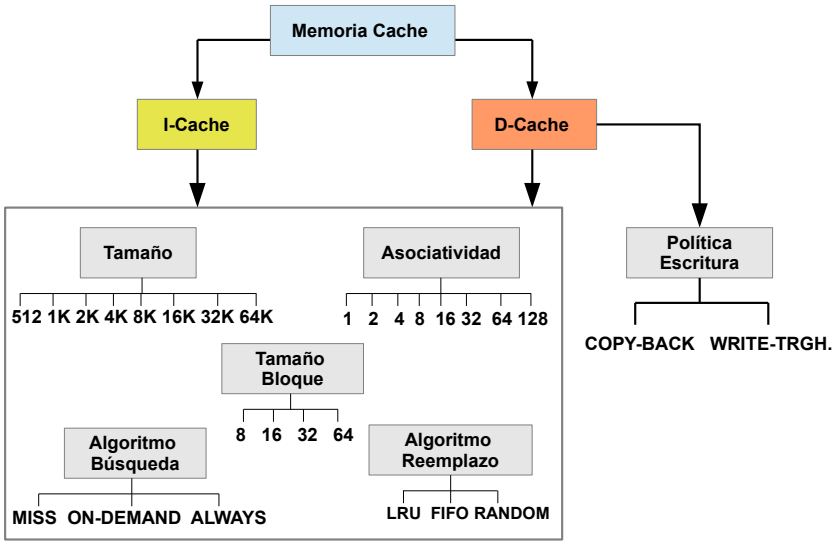


Figura 5.21: Parámetros de configuración y los posibles valores para cada uno de ellos. Cache de instrucciones y de datos deben ser parametrizadas con los valores disponibles.

Tabla 5.16: Configuración base de cache. Se forma con los parámetros de la cache de instrucciones y de datos.

Cache de Instrucciones				
Tamaño	**Tamaño de Bloque.**	**Asociatividad**	**Alg. Reemplazo**	**Alg. Búsqueda**
16 KB	32 B	4	LRU	On-Demand

Data Cache					
Tamaño	**Tamaño de Bloque.**	**Asociatividad**	**Alg. Reemplazo**	**Alg. Búsqueda**	**Pol. Escritura**
16 KB	32 B	4	LRU	On-demand	Copy-Back

En relación con la configuración cache de referencia, se toma como configuración base la utilizada en el enfoque propuesto para GE y que se muestra en la Tabla 5.16.

En primer lugar, la siguiente sección se dedica a analizar los resultados obtenidos utilizando el enfoque NSGA-II con el nuevo espacio de búsqueda y comparando estos nuevos resultados con los obtenidos previamente.

5.7.1. Resultados NSGA-II con espacios de búsqueda diferentes

Una vez llevado a cabo el proceso de optimización para NSGA-II con el nuevo espacio de búsqueda y la nueva configuración de cache base, las figuras 5.22, 5.23, 5.24, 5.25, 5.26 y 5.27 representan los frentes de Pareto obtenidos para cada una de

las aplicaciones objetivo. Cada punto, tal y como se ha mencionado anteriormente, representa al menos una configuración de cache que obtiene un valor para tiempo de ejecución y energía, calculados mediante las funciones objetivo ya descritas. Cabe destacar la diferencia considerable en el número de puntos representados en las aproximaciones a los frentes de Pareto, en comparación con aquellos obtenidos para el primer espacio de búsqueda. Esto es significativo dado que es posible garantizar la diversidad de las soluciones en la región de interés y evitar la convergencia a un único valor. Por ejemplo, *epic* presentaba un frente de Pareto con 3 puntos, mientras que ahora el frente de Pareto está compuesto por 13 puntos diferentes. Además, el conjunto de Pareto representado en dicho frente contiene varias configuraciones de cache para cada uno de los puntos de la gráfica.

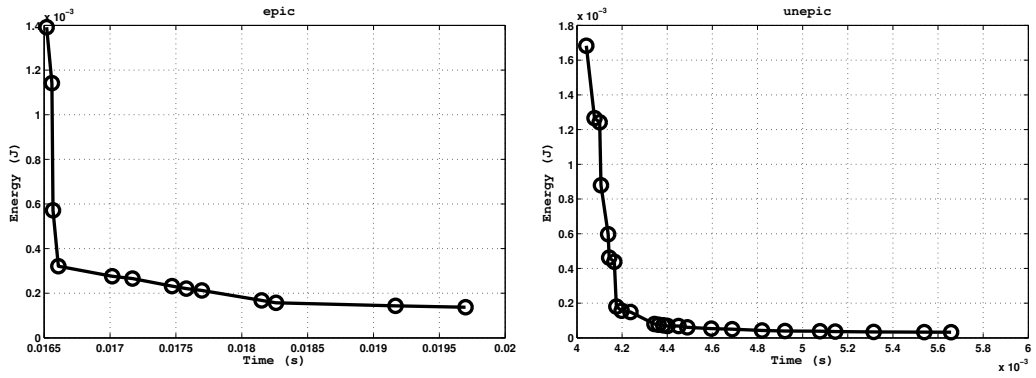


Figura 5.22: Representación del frente de Pareto para *epic*, *unepic*

Igual sucede con el resto de aplicaciones, con especial atención a aquellas aplicaciones donde aparecía un único punto en el frente, como sucedía para *pegwitdec*, *pegwitenc*, *mpegdec*, *rawcaudio* y *rawdaudio*. Sin embargo, estas aplicaciones de acuerdo a los resultados actuales presentan 12, 13, 17, 13 y 13 puntos, respectivamente. La ampliación que se produce en las aproximaciones a los frentes de Pareto es causa directa de usar un enfoque multi-objetivo puro y no una función de pesos.

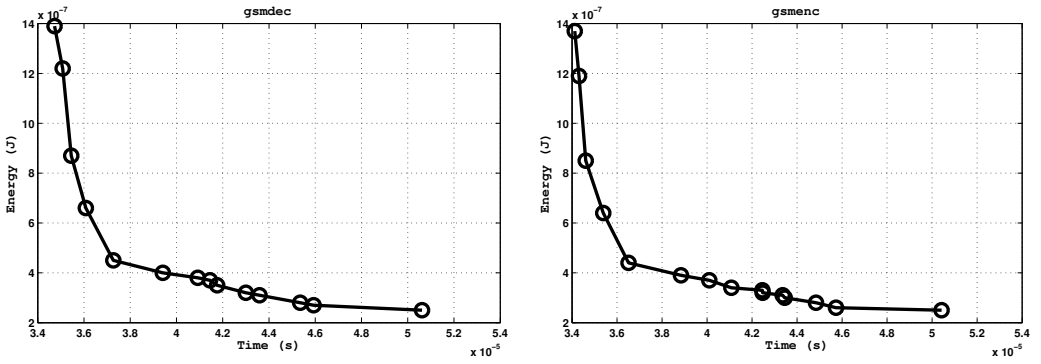


Figura 5.23: Representación del frente de Pareto para *gsmdec*, *gsmenc*

Con el fin de evaluar los resultados obtenidos con el nuevo espacio de búsqueda definido, la Tabla 5.17 muestra los porcentajes de mejora promedio en tiempo de ejecución y consumo de energía, para cada una de las soluciones del conjunto de Pareto de todas las aplicaciones objetivo. Comparando con los resultados alcanzados anteriormente y representados en la sección 5.5 en la Tabla 5.9, se observa como el porcentaje de mejora en tiempo de ejecución disminuye considerablemente, mientras que casi todas las aplicaciones mantienen un porcentaje de mejora alto en consumo de energía o incluso se incrementa. Uno de los motivos es el hecho de que la asociatividad sea de 64 en la primera configuración base de cache y de 4 en la actual. Tal y como se ha mencionado anteriormente, un nivel de asociatividad bajo permite reducir el tiempo de ejecución, pero incrementa el consumo energético.

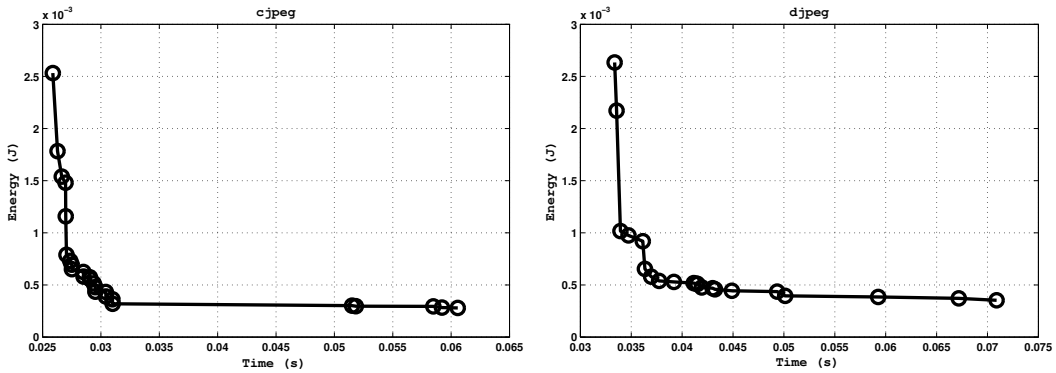


Figura 5.24: Representación del frente de Pareto para *cjpeg*, *djpeg*

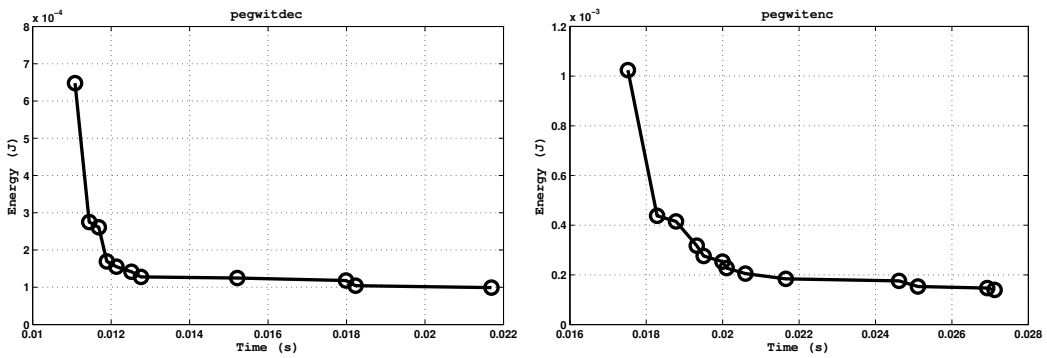


Figura 5.25: Representación del frente de Pareto para *pegwitdec*, *pegwitenc*

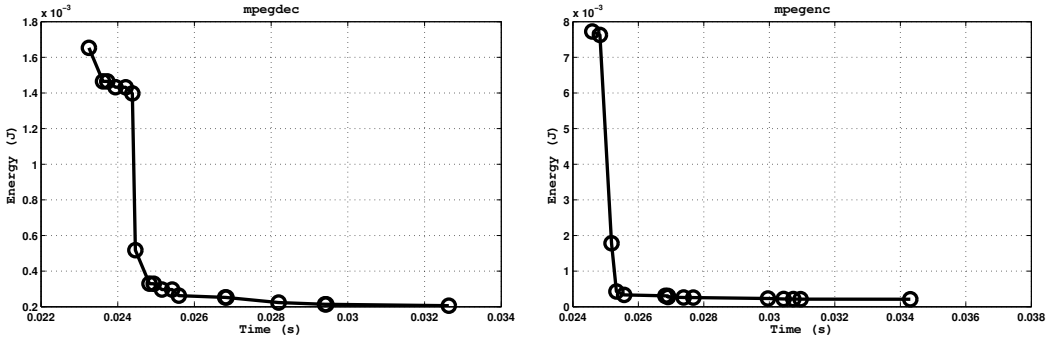


Figura 5.26: Representación del frente de Pareto para *mpegdec*, *mpegenc*

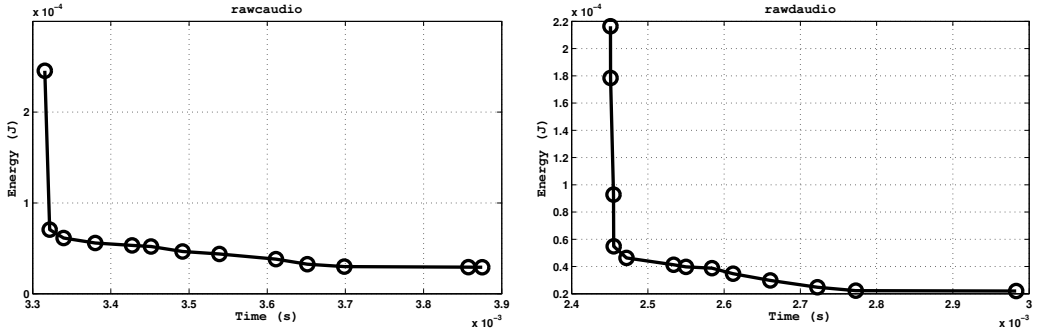


Figura 5.27: Representación del frente de Pareto para *rawcaudio*, *rawdaudio*

Por otra parte, generalmente, el incremento en el tamaño de bloque permite reducir el número de fallos y, en consecuencia, incrementar el rendimiento. Sin embargo, realizar este incremento de forma aislada y no combinado con la modificación de otros parámetros de cache, particularmente el tamaño de cache, puede llevar a que la reducción del número de fallos no sea significativa o incluso empeore. Esto se produce porque el número de bloques a almacenar en la cache se reduce,

hay mayor competitividad para situar un bloque en la cache y, en consecuencia, se incrementa la tasa de fallo y, por tanto, el tiempo de transferencia y la penalización por fallo. Teniendo en cuenta lo anterior, la diferencia entre las configuraciones de cache base, además de la asociatividad, es el tamaño de bloque que pasa de 16 Bytes a 32 Bytes.

Se hace necesario destacar el porcentaje de mejora en cuanto a consumo de energía de *unepic* y de *mpegenc*, dado que son las aplicaciones con los valores más bajos, en comparación al obtenido por el resto de aplicaciones. La respuesta viene dada por la conflictividad existente entre ambos objetivos, tiempo de ejecución y energía. Así, dentro de las soluciones en el conjunto de Pareto de *unepic* destacan 4 que presentan un porcentaje de mejora en tiempo de ejecución superior al 40 %, pero se produce un empeoramiento considerable (> 100 %), con respecto al consumo de energía. Para *mpegenc* aparecen también dos soluciones que en tiempo de ejecución presentan un porcentaje de mejora superior al 44 %, mientras que el empeoramiento del consumo de energía es superior al 200 %. Esto provoca que el porcentaje promedio de todas las soluciones del conjunto de Pareto se reduzca de forma importante. Si no se tuvieran en cuenta estas soluciones, el porcentaje de mejora en tiempo de ejecución y consumo de energía se situaría en el $30,97$ % y $83,18$ % para *unepic* y para *mpegenc* en el $36,28$ % y $83,58$ %. En consecuencia los porcentajes de mejora promedio para todas las aplicaciones, en tiempo de ejecución y consumo de energía, se situarían en $34,98$ % y $79,34$ %, respectivamente.

Sin embargo, incluso sufriendo una disminución importante en el porcentaje de mejora promedio respecto al tiempo de ejecución ($64,43$ % frente al $33,87$ %) y una diferencia más pequeña en cuanto al consumo de energía ($91,69$ % frente al $71,79$ %), el aspecto importante es que la metodología propuesta permite encontrar configuraciones que mejoran ambos objetivos en todas las aplicaciones abordadas.

Tabla 5.17: Porcentaje de mejora promedio de las soluciones del conjunto de Pareto y objetivo individual vs. la configuración base de cache, para todas las aplicaciones abordadas. La última fila representa el valor promedio de mejora de cada objetivo teniendo en cuenta todas las aplicaciones.

Aplicación	**Tiempo (%)**	**Energía (%)**
epic	44,91	76,29
unepic	33,42	26,74
gsmdec	31,38	84,31
gsmenc	18,97	83,59
cjpeg	27,18	71,07
djpeg	16,55	72,45
pegwitdec	27,07	83,87
pegwitenc	35,82	84,79
rawcaudio	48,76	84,84
rawdaudio	48,15	78,41
mpegdec	37,92	71,20
mpegenc	37,92	43,91
Promedio	33,87	71,79

5.7.2. Comparativa NSGA-II con GE

En esta sección se comparan los resultados obtenidos con el marco de optimización utilizando NSGA-II y Gramáticas Evolutivas. Para analizar estos resultados se han representado para cada aplicación los frentes de Pareto obtenidos con NSGA-II y las mejores soluciones alcanzadas mediante GE. El enfoque con GE utiliza una función de coste que pondera consumo de energía y tiempo de ejecución para evaluar cada configuración, por tanto para realizar la representación se han utilizado los valores individuales de tiempo de ejecución y consumo de energía, correspondientes a cada una de las mejores soluciones para GE.

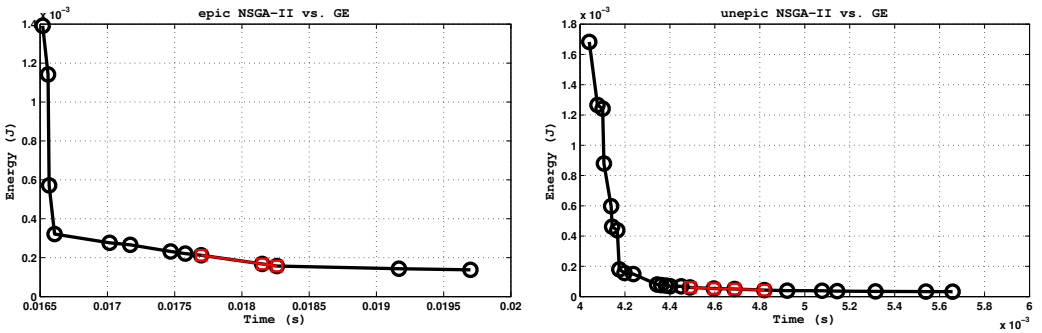


Figura 5.28: Representación del frente de Pareto para NSGA-II (en color negro), y valores objetivo de las mejores soluciones obtenidas con GE (en color rojo), para *epic* y *unepic*.

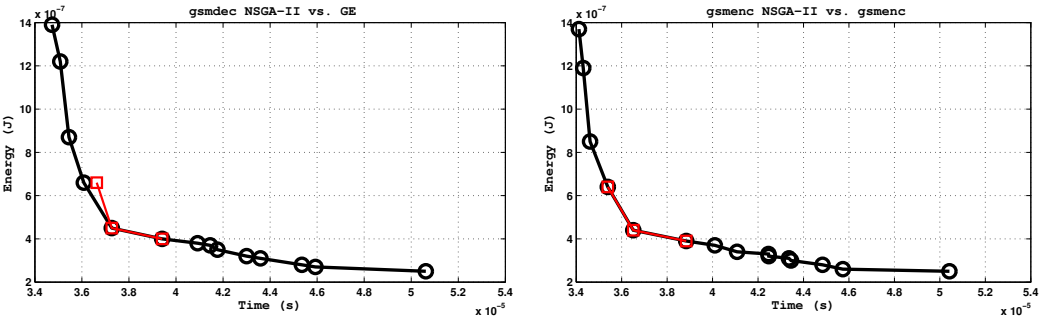


Figura 5.29: Representación del frente de Pareto para NSGA-II (en color negro), y valores objetivo de las mejores soluciones obtenidas con GE (en color rojo), para *gsmdec* y *gsmenc*.

De esta forma, las figuras 5.28, 5.29, 5.30, 5.31, 5.32 y 5.33 muestran los frentes de Pareto obtenidos por NSGA-II en color negro y los valores objetivo correspondientes a las mejores soluciones con GE en color rojo. En general, para todas las aplicaciones NSGA-II proporciona un conjunto de Pareto que cubre una región mayor que las soluciones aportadas por GE. Respecto a la uniformidad en la distribución de las soluciones de NSGA-II en el espacio objetivo, algunas de las aplicaciones como *epic*, *unepic*, *gsmdec* y *gsmenc* presentan una uniformidad mayor que el resto, pero todas aportan un número de soluciones elevado, en comparación con las soluciones de GE. GE aporta muy buenas soluciones compatibles con el frente de Pareto de NSGA-II, incluso para *mpegenc* ambas metaheurísticas coinciden en la mejor solución encontrada. Sin embargo, GE obtiene un número de soluciones considerablemente menor a NSGA-II y la mayoría de las soluciones se encuentran

en un área determinada del espacio de la función objetivo. Esto se debe a la naturaleza de la búsqueda en GE con la utilización de una función ponderada que tiende al punto medio del frente.

La Figura 5.28 muestra los frentes de Pareto para NSGA-II y los valores objetivo de las mejores soluciones proporcionadas por GE, para *epic* y *unepic*. Las soluciones de GE, en ambas aplicaciones, pertenecen a la aproximación al frente de Pareto aunque presentan poca diversidad y se sitúan en un área reducida del espacio de la función objetivo que se identifican por un consumo bajo de energía. Sin embargo, se alejan del mejor valor proporcionado por NSGA-II como consecuencia de los valores en tiempo de ejecución.

La representación para *gsmdec* y *gsmenc* se muestra en la Figura 5.29. Aunque GE presenta poca diversidad, las soluciones aportadas por GE son también las mejores soluciones encontradas por NSGA-II. Sólo una de las soluciones obtenidas por GE para *gsmdec* es claramente dominada por las soluciones representadas en la aproximación al frente de Pareto de NSGA-II.

La Figura 5.30 representa gráficamente los frentes de Pareto para NSGA-II y los valores objetivo de las mejores soluciones proporcionadas por GE, para *cjpeg* y *djpeg*. El conjunto de soluciones aportadas por NSGA-II presenta de forma clara mejor diversidad, uniformidad y explotación del espacio de la función objetivo. Las soluciones obtenidas por GE son buenas soluciones y excepto una de las soluciones en *cjpeg*, todas pertenecen también a la aproximación al frente de Pareto de NSGA-II y se sitúan en el área donde se representan las mejores soluciones. Sin embargo, las soluciones de GE presentan poca diversidad y se encuentran localizadas en un área muy reducida del espacio de la función objetivo.

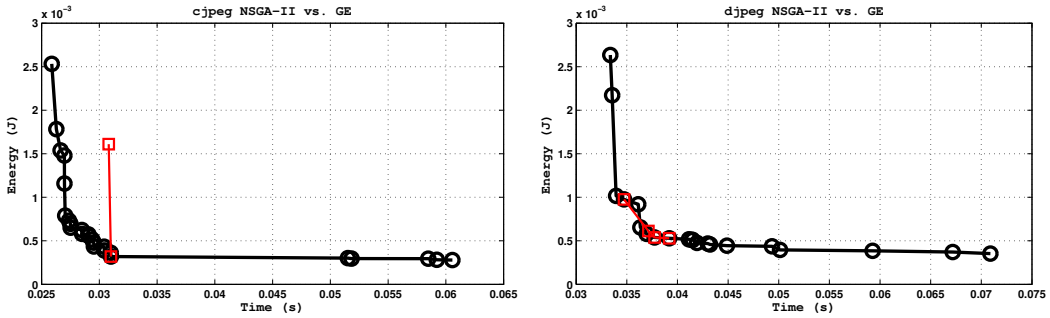


Figura 5.30: Representación del frente de Pareto para NSGA-II (en color negro), y valores objetivo de las mejores soluciones obtenidas con GE (en color rojo), para *cjpeg* y *djpeg*.

Los resultados obtenidos para las aplicaciones *pegwitdec* y *pegwitenc* se muestran en la Figura 5.31. Al igual que las aplicaciones anteriores NSGA-II explota mejor el espacio de la función objetivo. Las soluciones de GE, a pesar de presentar poca diversidad, coinciden con las mejores soluciones encontradas por NSGA-II. De forma similar, las soluciones que se han obtenido para las aplicaciones *mpegdec* y *mpegenc*, utilizando ambas metaheurísticas, se representan en la Figura 5.32. Se mantiene el mismo comportamiento, donde las soluciones de GE se concentran en un área reducida de la función objetivo, pero muy cercano o coincidente con las mejores soluciones obtenidas por NSGA-II.

La representación de las soluciones para las aplicaciones *rawcaudio* y *rawdaudio* se muestran en la Figura 5.33. A excepción de una de las soluciones en ambas aplicaciones, todas pertenecen a la aproximación al frente de Pareto de NSGA-II y, aunque no coinciden con las mejores soluciones aportadas por NSGA-II, las soluciones obtenidas por GE también son buenas soluciones.

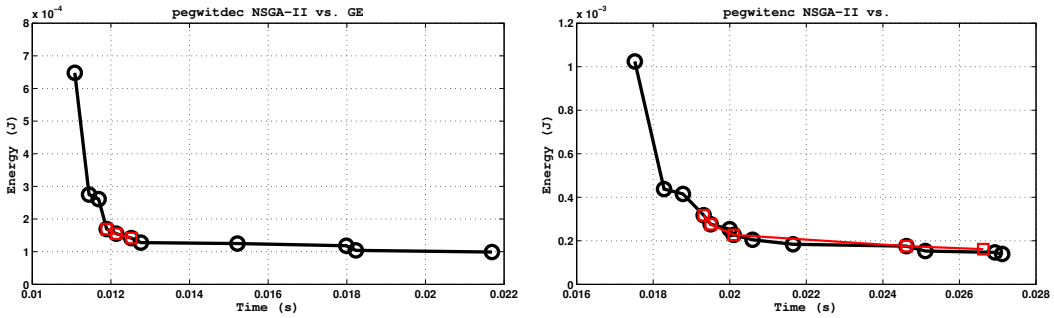


Figura 5.31: Representación del frente de Pareto para NSGA-II (en color negro), y valores objetivo de las mejores soluciones obtenidas con GE (en color rojo), para *pegwitdec* y *pegwitenc*.

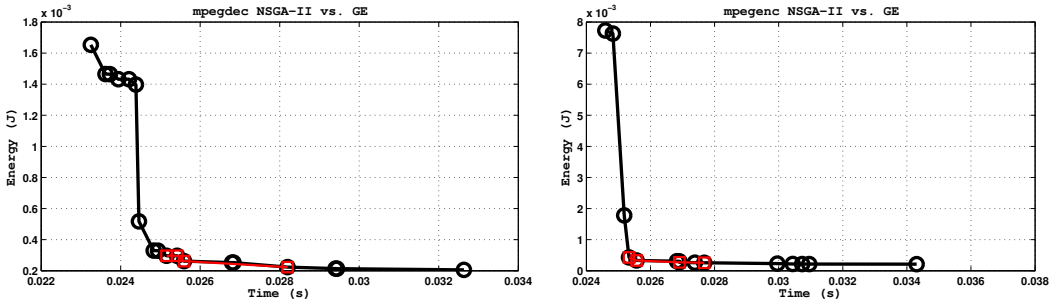


Figura 5.32: Representación del frente de Pareto para NSGA-II (en color negro), y valores objetivo de las mejores soluciones obtenidas con GE (en color rojo), para *mpegdec* y *mpegenc*.

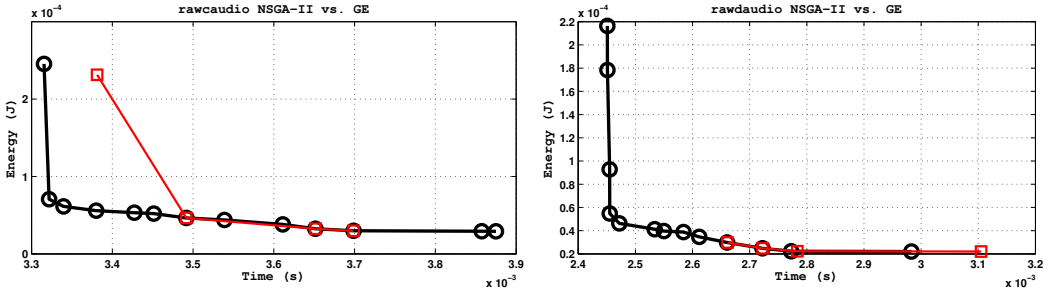


Figura 5.33: Representación del frente de Pareto para NSGA-II (en color negro), y valores objetivo de las mejores soluciones obtenidas con GE (en color rojo), para *rawcaudio* y *rawdaudio*.

Además, la representación gráfica tanto de la aproximación al frente de Pareto de NSGA-II y de las soluciones obtenidas por GE, hace posible el análisis de los valores objetivo a minimizar. Así se observa como la mayoría de las soluciones, aportadas por el marco de optimización con GE, se encuentran dentro de aquellas

con un consumo de energía menor, aunque no así en cuanto a tiempo de ejecución. GE encontraría un óptimo local para cada aplicación.

A la vista de los resultados obtenidos, se confirma que NSGA-II explota mejor el espacio objetivo, proporciona mayor diversidad y presenta una distribución de las soluciones en el espacio objetivo más uniforme. No se puede afirmar que el frente de Pareto obtenido mediante NSGA-II sea el verdadero POF, pero de acuerdo a los resultados alcanzados y corroborados en algunas de las mejores soluciones mediante GE, es probable que se aproxime bastante a él, garantizando de esta manera una buena convergencia de las soluciones. Por último, es importante destacar cómo GE compite bien con NSGA-II dado que la mayoría de las soluciones obtenidas forma parte del frente en cada una de las aplicaciones objetivo. Por otra parte, en las aplicaciones *gsmdec* y *cjpeg* aparecen soluciones obtenidas por GE que son claramente dominadas por NSGA-II.

Para comprobar cuánto de buenos son los resultados obtenidos respecto de la configuración de cache utilizada como referencia se ha realizado el test estadístico *t-student* que permite comparar las medias con un valor estándar y que se detalla en la siguiente sección.

5.7.3. Cálculo estadístico

Los últimos resultados, como se observa en la Tabla 5.17, indican que utilizando la metodología propuesta es posible encontrar configuraciones de cache para todas las aplicaciones abordadas, que en promedio mejoran el 33, 87 % y el 71, 79 % respecto al tiempo de ejecución y el consumo de energía, respectivamente. Con el fin de corroborar la naturaleza significativa de los resultados alcanzados, se ha realizado el test estadístico mediante el cálculo del *p-valor*[3] que permite determinar la probabilidad de que los resultados de una prueba sean o no significativos. Para llevarlo a cabo se ha utilizado el software estadístico *R* [187].

[3]El valor p, que determina si los resultados son estadísticamente significativos. El valor p se sitúa entre 0 y 1 y los valores más bajos de probabilidad sustentan la decisión de rechazar la hipótesis nula.

Tabla 5.18: Cálculo de p-valor para los conjuntos de Pareto de todas las aplicaciones frente a una configuración cache de referencia. *DF* representa los grados de libertad, *T* corresponde al estadístico de contraste y *p-valor* es la probabilidad que mide la significatividad.

		Tiempo		**Energía**	
Aplicación	**DF**	**T**	**p-valor**	**T**	**p-valor**
cjpeg	24	-5,47	6,42E-006	-15,87	1,59E-014
djpeg	19	-3,58	1,01E-003	-14,48	5,10E-009
epic	12	-50,71	1,13E-012	-11,61	3,48E-005
unepic	26	-23,79	2,20E-016	-1,23	1,15E-001
unepic*	23	-22,09	2,20E-016	-5,52	6,41E-003
gsmdec	16	-17,66	3,23E-012	-32,69	2,20E-016
gsmenc	14	-7,59	1,27E-003	-28,48	4,28E-011
mpegdec	16	-25,96	8,30E-012	-11,38	2,20E-006
mpegenc	15	-24,21	9,77E-011	-1,60	6,50E-002
mpegenc*	13	-22,93	3,36E-009	-17,61	9,38E-008
pegwitdec	11	-4,53	4,31E-004	-22,55	7,35E-008
pegwitenc	12	-13,09	9,15E-006	-25,81	3,49E-009
rawcaudio	12	-63,21	2,20E-016	-21,43	3,11E-008
rawdaudio	12	-54,49	4,81E-013	-13,65	5,70E-006

El cálculo se ha realizado con el conjunto de Pareto obtenido en este último experimento y la configuración cache de referencia. Los resultados se muestran en la Tabla 5.18, donde **DF** es el grado de libertad [4]. **T** corresponde al valor estadístico que se utiliza para hacer la comparación o estadístico de contraste [5] y **p-valor** es el valor que permite rechazar que el valor medio de los datos es el estándar o valor con el que se compara. Se admite que es menor que este si **p-valor** es inferior al $0,05$.

En todas las aplicaciones el grado de libertad corresponde al número de soluciones que proporciona cada conjunto de Pareto, excepto para las aplicaciones unepic y mpegenc, donde el grado de libertad se ha reducido a 23 y 13, respectivamente. Ya hemos visto en la sección anterior que dentro del conjunto de Pareto para estas dos

[4]Grado de libertad es el número de valores elegidos libremente en una muestra y que permiten llegar a un valor.

[5] Estadístico de Contraste es una variable aleatoria con una distribución muestral que proporciona un conjunto de probabilidades para un valor o intervalo de valores. Se utiliza para realizar la toma de decisión en un contraste de hipótesis.

aplicaciones, hay varias soluciones (4 para unepic y 2 para mpegdec) que si bien obtienen una buena mejora en tiempo de ejecución, empeoran significativamente en consumo de energía. Por esta motivo, las líneas correspondientes a unepic* y mpegenc* realizan el cálculo de p-valor sin tener en cuenta estos valores que desvirtúan el valor medio.

En la tabla se observa que el **p-valor** es muy inferior a $0,05$ para las 12 aplicaciones en cuanto al rendimiento y consumo de energía. En vista a los datos del test estadístico, podemos decir que los resultados alcanzados por la metodología de optimización propuesta, para la que se ha medido el valor **p**, son significativos.

En esta tesis se presenta la optimización de tres niveles del subsistema de memoria, el banco de registros, el subsistema de memoria cache y la gestión de memoria dinámica. El banco de registros se ha abordado en el Capítulo 4 y el subsistema de memoria cache se ha acometido en el Capítulo 5. A continuación, en el siguiente capítulo se aborda la gestión de memoria dinámica que es el tercer nivel del subsistema de memoria objeto de optimización en esta tesis.

Capítulo 6

Optimización de la Memoria Dinámica

6.1. Introducción

En los actuales dispositivos móviles la ejecución de aplicaciones multimedia que presentan una alta tasa de utilización de recursos, ocupan un lugar predominante. Actualmente, este tipo de aplicaciones se ejecutan frecuentemente en una amplia variedad de plataformas hardware como son ordenadores personales, consolas de videojuegos, teléfonos inteligentes y otros dispositivos móviles. Además, las aplicaciones multimedia suelen llevar a cabo gran cantidad de operaciones de memoria dinámica debido a la naturaleza inherente de los datos que manejan. La asignación y liberación de memoria dinámica, por ejemplo, en C++ se realiza utilizando los operadores *new()* y *delete()* que se mapean a las funciones *malloc()* y *free()* de C estándar. El Gestor de Memoria Dinámica (*DMM*), también conocido como *memory allocator*, es quien se encarga de gestionar la memoria dinámica y es un componente crucial que influye tanto en el comportamiento general de la aplicación como en el comportamiento hardware. El uso de memoria dinámica puede mejorar el tiempo de ejecución y el uso de memoria de un gran número de aplicaciones [188, 189]. Sin embargo, usar un DMM especialmente dirigido a una aplicación específica puede mejorar considerablemente el comportamiento de la aplicación [190, 191]. El enfoque que muchos programadores llevan a cabo, para alcanzar el rendimiento más alto, es el de diseñar sus propios gestores de memoria como macros o funciones

monolíticas con el fin de evitar la sobrecarga de llamadas a función. Este método está considerado como dentro de las mejores prácticas para programadores cualificados [192]. Sin embargo, esta metodología está asociada a la dificultad del desarrollo de gestores de memoria, al gran esfuerzo necesario para su mantenimiento, la necesidad de modificación si se desea utilizar para otra aplicación distinta o la aplicación se modifica y, por tanto, la alta probabilidad de error en cualquiera de estas fases.

En este sentido, la optimización del subsistema de memoria dinámica se considera una tarea necesaria con el objetivo puesto en que las aplicaciones multimedia se ejecuten de forma eficiente en cualquier tipo de sistemas, incluyendo los sistemas empotrados. Esta es una tarea compleja que se puede abordar diseñando nuevos mecanismos de gestión de memoria dinámica. En los últimos años, se han propuesto varias metodologías automáticas para optimizar gestores de memoria dinámica personalizados. De esta forma, es prioritario dar respuesta a la pregunta clave de cómo medir la influencia de un DMM en el comportamiento de la plataforma hardware y de las aplicaciones. Muchos de los enfoques propuestos se dirigen principalmente a mejorar el rendimiento de una aplicación, a diferencia del enfoque que se presenta en esta tesis. En esta tesis se propone una nuevo método de optimización basado en GE que aborda el diseño de DMM personalizados para una aplicación dada, minimizando el uso de la memoria y maximizando el rendimiento. Este método permite la evaluación de cada DMM de acuerdo a la aplicación y plataforma hardware en la que se ejecuta, sin necesidad de modificar la aplicación objetivo.

En el Capítulo 2, sección 2.2 se han mencionado algunas de las propuestas realizadas en este campo. A continuación las mencionamos brevemente y presentamos las principales diferencias con la propuesta que aquí se realiza.

Berger et al. en [3] diseñan una biblioteca de plantillas desarrolladas en C++ que permite implementar diferentes gestores de memoria y estudiar el comportamiento de gestores de memoria dinámica personalizados y de propósito general, para programas C++. Como métricas se utilizan el tiempo de ejecución y uso de la

memoria. Los resultados que presentan son bastante representativos, sin embargo es necesario un gran esfuerzo de codificación para extenderlos a otras aplicaciones o diseños de gestores de memoria. Además, en este estudio no se tienen en cuenta métricas tan significativas como el consumo de energía y la temperatura, que son verdaderamente importantes, especialmente para dispositivos móviles.

Lo et al. en [84] presentan un DMM que recibe trazas provenientes de programas desarrollados en Java y C++ y realiza su simulación, de acuerdo a un esquema de asignación seleccionado por el usuario. El trabajo proporciona algunas técnicas para generar trazas y el simulador presenta un conjunto de métricas relativas al tiempo de ejecución y uso de la memoria. Teng et al. en [85] presentan un enfoque similar. Sin embargo, estos simuladores limitan a un conjunto reducido de esquemas de asignación y liberación de memoria, y el mejor DMM se selecciona en base a un conjunto de reglas predefinidas y búsqueda mono-objetivo. Además, tampoco se analiza el consumo energético y la temperatura.

Del Rosso en [86] evalúa el rendimiento de diferentes DMMs para sistemas empotrados en tiempo real. Este trabajo se centra en el estudio de la fragmentación de memoria, que es la principal medida para los sistemas empotrados en tiempo real. Además, estudia una métrica independiente de la CPU y distinta al tiempo de ejecución, denominada *performance speed metric*. Esta métrica evita la influencia del código de instrumentación en las restricciones de tiempo real. No obstante, este estudio se limita a sistemas empotrados en tiempo real y también adolece de medidas tan importantes como el consumo de energía y temperatura.

Más recientemente, Atienza et al. en [87] y [88] proponen un método para evaluar la memoria y la energía consumida por un DMM. Sin embargo, es necesario implementar e integrar el DMM en la aplicación objetivo. Además, estos enfoques propuestos requieren que la aplicación sea ejecutada cada vez que se quiere evaluar un gestor de memoria personalizado, y esta tarea consume mucho tiempo, particularmente si la aplicación necesita de interacción con el usuario, como sucede con los videojuegos.

Ante la patente necesidad de abordar el diseño de DMM dedicados mediante técnicas multi-objetivo, Risco et al. [89] proponen un algoritmo evolutivo multi-objetivo paralelo para optimizar aplicaciones típicamente ejecutadas en sistemas de sobremesa, para su uso en sistemas empotrados multimedia. El objetivo era mejorar el proceso de asignación de tipos de datos dinámicos (DDT, *Dynamic Data Type*), que afecta al rendimiento. Este trabajo mejora el rendimiento medio, uso de la memoria y consumo de energía del subsistema de memoria. A partir del anterior trabajo, Risco et al. en [90] presentan una primera propuesta de un algoritmo de optimización basado en GE, denominado GEA, con el fin de diseñar DMM personalizados y en [91, 92] se presenta y describe el simulador DMM que se utiliza y se ha modificado para obtener los objetivos de esta tesis. Sin embargo, el espacio de diseño se define según las clasificaciones realizadas en [93] y [87]. Esto conlleva un espacio de diseño muy grande y el proceso de clasificación da lugar a una taxonomía compleja de los DMM. Además, el tiempo de ejecución necesario para ejecutar el algoritmo y el simulador lo convierte en inaccesible y en [94] se presenta una versión paralela. Sin embargo, a diferencia de las propuesta que se presentan en esta tesis, el perfilado de las aplicaciones se realiza sobrecargando las funciones *malloc*() y *free*() que requiere modificar y re-compilar cada aplicación objetivo y este proceso consume mucho tiempo.

En esta tesis se presenta una metodología basada en gramáticas evolutivas, para diseñar gestores de memoria dinámica a medida de una aplicación objetivo. Esta metodología utiliza el perfilado estático de las aplicaciones para extraer las características de la aplicación que serán utilizadas tanto para el proceso de diseño de la gramática BNF como para evaluar cada DMM diseñado. La Figura 6.1 muestra gráficamente las diferentes fases de los objetivos propuestos.

El marco de optimización propuesto combina el uso de la herramienta de instrumentación Pin [193] y el simulador DMM, anteriormente mencionado. El método se ha probado con seis aplicaciones pertenecientes al conjunto de aplicaciones SPEC [194] con algunas de sus entradas estándar y cinco DMM de propósito gene-

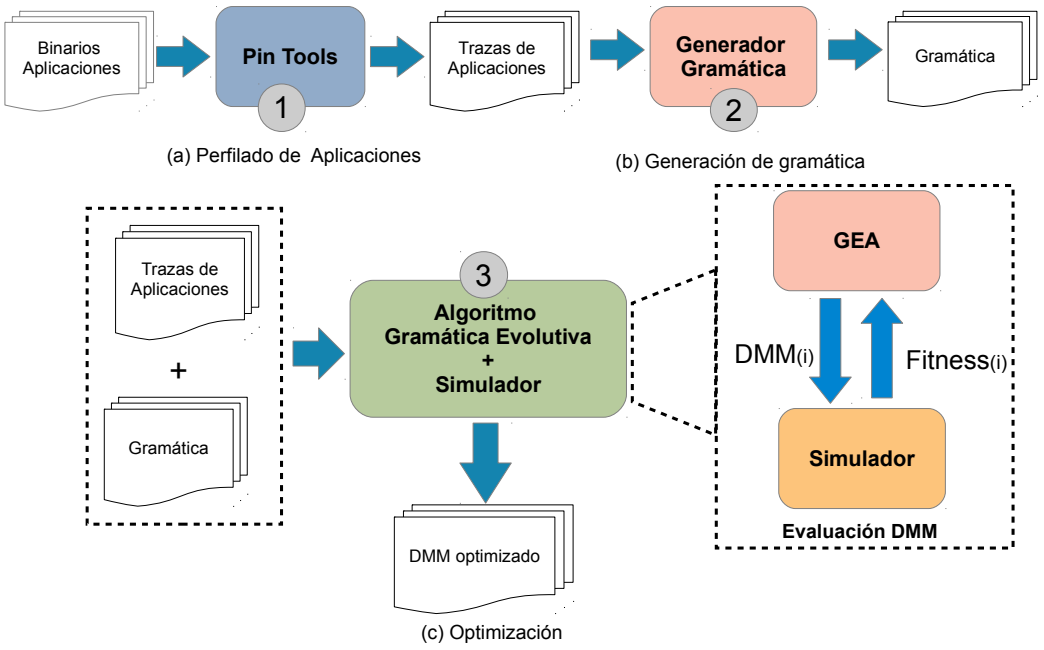


Figura 6.1: Metodología utilizada en el marco de optimización con GE. Estructurado en tres fases, la primera es responsable de obtener las trazas de las aplicaciones mediante la herramienta Pin; la segunda tiene como objetivo el diseño de la gramática personalizada por aplicación y la tercera fase se encarga del algoritmo de optimización con GE que llama a su vez al simulador para evaluar cada individuo con todos y cada uno de los DMM abordados y que corresponde a la metodología propuesta como objetivo (1).

ral. Los resultados muestran que esta técnica simplifica la tarea de los diseñadores en la evaluación de DMM multi-capa y aporta mejoras al diseño de DMM para una aplicación objetivo. Además este método permite reducir el tiempo de ejecución y el uso de la memoria en un $59,27\,\%$ de media, al comparar con el valor objetivo global.

6.2. Gestión dinámica de memoria

Esta sección describe y explica brevemente los fundamentos de la gestión de memoria dinámica y el funcionamiento del simulador DMM anteriormente mencionado y utilizado en esta tesis.

La utilización de la memoria dinámica permite que los programas hagan mejor uso de la memoria del sistema, reservando únicamente la memoria que van a utilizar. La memoria dinámica almacena los bloques de memoria que se crean y destruyen

en tiempo de ejecución. El software responsable de tratar con estos bloques es el gestor de memoria dinámica. El gestor de memoria es, por tanto, un algoritmo que lleva el control en tiempo real del uso de la memoria y qué partes se encuentran libres y cuáles ocupadas. El DMM recibe las peticiones de asignación de bloques de memoria y las peticiones de cancelación de las asignaciones realizadas para liberar los bloques de memoria. De esta forma, es capaz de mejorar el comportamiento de la aplicación mediante la utilización de los bloques libres, y así reducir el número de llamadas al sistema para peticiones nuevas de memoria. Además el gestor de memoria permite minimizar los problemas debidos a la gestión ineficiente de la memoria como es el problema de la fragmentación (externa e interna), con un consumo de tiempo razonable. La fragmentación interna aparece cuando la memoria asignada es mayor que la que se necesita. La fragmentación externa se da cuando hay suficiente memoria disponible distribuida en huecos de diferente tamaño, no contiguos y no se puede asignar memoria a una petición.

Para solucionar el problema de la fragmentación, el DMM puede utilizar mecanismos adicionales como la unión de dos bloques de memoria (*coalescing*) o la división de un bloque en dos partes (*splitting*), con el objetivo de mejorar el aprovechamiento de memoria y dar respuesta a las peticiones de asignación de memoria de un tamaño determinado y minimizar, sobre todo, la fragmentación interna. Sin embargo, estos mecanismos también suponen una sobrecarga debido a que los nuevos bloques obtenidos mediante estas estrategias deben ubicarse en listas de bloques libres diferentes a la que se encontraban. Además, los DMM normalmente consumen más memoria que la memoria solicitada por las aplicaciones debido a los rangos disponibles de tamaños de bloque y las estructuras de datos necesarias para alojarlos y gestionarlos [195].

El DMM trabaja como se indica a continuación. Una vez que el DMM recibe una llamada de liberación de memoria, el DMM marca el bloque de memoria como libre y lo almacena en una lista ordenada dispuesto para ser reutilizado posteriormente. Por otra parte, después de recibir una llamada de asignación de memoria, el DMM

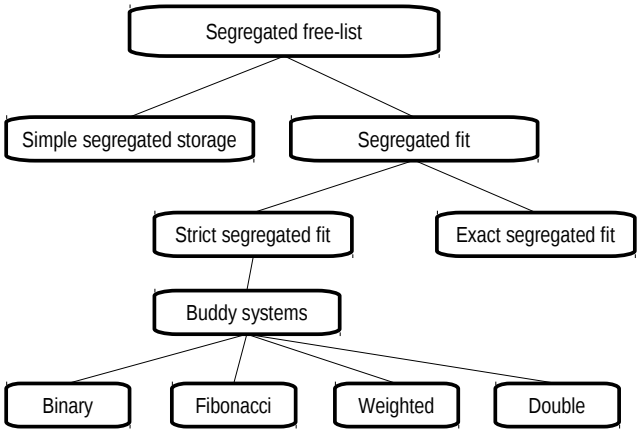


Figura 6.2: Clasificación de los gestores de memoria según [196].

busca en las listas de bloques libres. Si el DMM no encuentra ningún bloque que encaje con la petición, entonces solicita mediante una llamada al sistema memoria nueva y el sistema operativo devuelve un bloque nuevo de memoria.

Un DMM se puede implementar utilizando una o más estructuras de datos para almacenar bloques libres de memoria, que son gestionadas utilizando diferentes políticas y mecanismos de asignación de memoria. La Figura 6.2 muestra la clasificación que según [196] representa a la mayoría de ellos y que se explica brevemente a continuación.

Segregated free-list allocator (gestor diferenciado de la lista de bloques libres), divide la lista de bloques libres en varios subconjuntos, en base al tamaño de los bloques libres. El proceso se realiza mediante indexado de la lista de bloques libres diferenciado por tamaño de bloque, no en cuanto a almacenamiento físico diferente. La asignación y liberación de los bloques se resuelven buscando la lista más adecuada según el tamaño solicitado. Este tipo de mecanismo generalmente implementa la política del ajuste perfecto o mejor ajuste.

Simple segregated storage (almacenamiento diferenciado sencillo) es un mecanismo de asignación diferenciado de la lista de bloques libres que divide el almacenamiento en áreas, asignando objetos de un único tamaño, o de un pequeño rango de tamaños, dentro de cada área. Este enfoque realiza la asignación de forma rápida y evita las cabeceras, pero puede llevar a un alto nivel de fragmentación externa,

como son aquellas partes no utilizadas de zonas que no pueden ser reutilizadas para otros tamaños de objeto. Este mecanismo no realiza división de bloques, así ante una petición de un tamaño para el que no hay bloques libres, se realiza una solicitud al sistema operativo que sirve las páginas de memoria y las divide para servir la petición de acuerdo al tamaño solicitado.

Segregated fit (ajuste diferenciado), es otra variación del mecanismo de asignación de la lista de bloques libres diferenciado. En esta variación, cada lista gestiona los bloques libres de un rango de tamaños determinado. El gestor identifica la lista de bloques libres adecuada y realiza la asignación de la petición correspondiente (habitualmente utilizando la política del primer ajuste). Si el mecanismo de asignación falla, se toma un bloque mayor desde otra lista dividiéndola según corresponda. Si no existe un bloque mayor se realiza la petición al Sistema Operativo.

Strict segregated fit (ajuste estricto diferenciado), es un mecanismo de asignación de ajuste diferenciado que tiene un único tamaño de bloque en cada lista de bloques libres. La petición de un tamaño de bloque se redondea al siguiente tamaño inmediatamente superior al solicitado y se devuelve el primer bloque de dicha lista. Los tamaños se deben seleccionar de modo que cualquier bloque de un tamaño más grande se pueda dividir en un número de bloques de tamaño más pequeño.

Buddy systems (sistemas de amigos), son casos especiales de asignación de ajuste diferenciado estricto que realiza las operaciones de división y fusión de forma rápida emparejando cada bloque con un único bloque amigo adyacente. En este sentido, se dispone de una serie de listas de bloques libres y cada una de las listas contiene uno de los tamaños permitidos de bloque. La asignación redondea el tamaño solicitado al tamaño inmediatamente superior permitido y se realiza la asignación desde la lista de bloques libres correspondiente. Si la lista de bloques libres está vacía, se puede seleccionar un bloque de mayor tamaño y dividirlo. Un bloque sólo puede ser dividido en un par de bloques amigos y un bloque sólo puede ser fusionado con su amigo, y esto es sólo posible si el amigo no ha sido dividido en bloques más pequeños. Los diferentes tipos de sistemas de amigos se diferen-

cian por los tamaños de bloque disponibles y el método utilizado de división. Estos incluyen amigos binarios (*binary buddies*), que es el tipo más habitual, amigos de Fibonacci (*Fibonacci buddies*), amigos ponderados (*weighted buddies*) y dobles amigos (*double buddies*).

Exact segregated fit (ajuste exacto diferenciado), es un gestor de memoria de ajuste diferenciado que tiene una lista separada de bloques libres para cada tamaño de bloque posible. El conjunto de listas de bloques libres se puede representar de forma dispersa. Los bloques grandes se pueden tratar de forma separada. Los detalles del mecanismo dependen de la distribución de tamaños entre las listas de bloques libres.

Aparte de esta distribución, existen varios DMMs de propósito general. A continuación se describen brevemente dos de ellos, el DMM Kingsley [196] y el DMM Lea [189], por ser representativos de los sistemas Windows y Linux, respectivamente. Kingsley y Lea son ampliamente utilizados tanto en sistemas empotrados como en sistemas de propósito general. El DMM Kingsley fue utilizado originalmente en BSD 4.2 y los actuales sistemas basados en Windows (tanto plataformas móviles como de escritorio), aplican las principales ideas de Kingsley. El DMM Kingsley organiza la memoria disponible en tamaños de bloque, potencia de dos: todas las peticiones de asignación se redondean al siguiente tamaño potencia de dos. La asignación se lleva a cabo sin realizar división (hacer bloques más pequeños a partir de bloques de mayor tamaño) o fusión (combinar bloques libres adyacentes). Este algoritmo es bien conocido por ser uno de los DMM más rápidos, aunque también se encuentra entre los peores en términos de uso de la memoria [93]. El objetivo principal en el DMM Kingsley es la velocidad, que consigue al asignar o liberar las peticiones realizadas, así la prevención de la fragmentación es un objetivo secundario.

Por el contrario, el DMM Lea es una aproximación al DMM del mejor ajuste que presenta principalmente una utilización baja de memoria y minimiza el problema de la fragmentación, manteniendo una velocidad razonable. Los sistemas basados en

Linux utilizan como base el DMM Lea. Lea presenta un comportamiento diferente basado en el tamaño de la memoria solicitada. Por ejemplo, a los objetos pequeños (menores de 64 bytes), se les asignan los bloques libres utilizando una política de ajuste exacto (una lista enlazada de bloques por cada múltiplo de 8 bytes). Para los objetos de mediano tamaño (menos de 128 Kb), el gestor Lea realiza fusión y división inmediata de bloques en las listas anteriores y utiliza la aproximación del mejor ajuste. Para tamaños grandes ($\geq$ 128 Kb), se utiliza memoria virtual (mediante la llamada al sistema *mmap*).

En el marco de optimización propuesto con GE, cada DMM se compara con los DMM Kingsley y Lea que tienen como objetivos maximizar el rendimiento y minimizar el uso de la memoria, respectivamente.

Simulador DMM

Actualmente, existen varias librerías especialmente dedicadas a la implementación de DMMs personalizados y de propósito general [197, 189, 93]. Cuando se utilizan estas librerías, cada DMM debe ser implementado, compilado y validado contra una aplicación objetivo. Por tanto, aunque una librería DMM determinada tenga un alto nivel de modularidad, este es un proceso que consume mucho tiempo. Para enfrentarse a estos cambios, se ha desarrollado un simulador DMM que permite al diseñador de sistemas evaluar, sin ningún esfuerzo adicional para el diseñador, el impacto de un DMM en un sistema dado para una aplicación objetivo [198].

El diseño global de la plataforma software lo constituye una librería de simulación de DMM, una librería de optimización (basada en GE) y una Interfaz Gráfica de Usuario (GUI, Graphical User Interface) que tiene implementados los gestores de memoria Kingsley y Lea disponible en [199]. En la Figura 6.3 se puede ver la interfaz gráfica del simulador que facilita el uso del simulador, para evaluar algunos gestores de memoria dinámica de propósito general, así como para realizar la exploración automática de los DMM para una aplicación objetivo. Dada una traza de programa, la interfaz simula el gestor seleccionado, proporcionando su esquema

y alguna de las métricas calculadas. Todas las métricas se almacenan en ficheros externos.

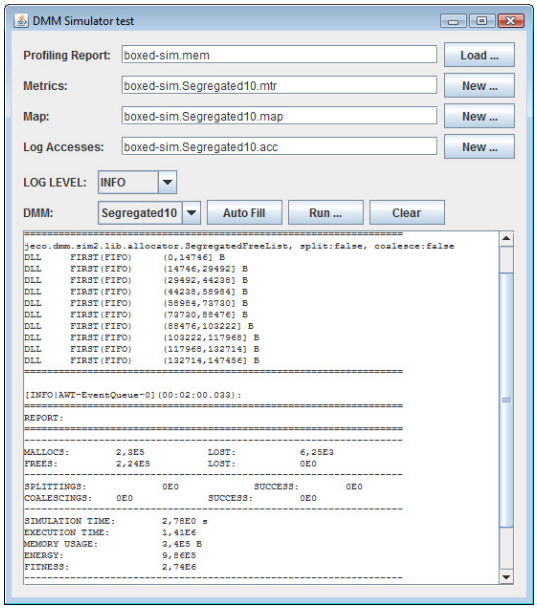


Figura 6.3: Interfaz gráfica de usuario del simulador DMM.

Para trabajar con este simulador, el diseñador comienza con una fase que se encarga de generar la traza de la aplicación objetivo. Con este fin, en esta tesis se ha utilizado la herramienta Pin que permite la inserción de las llamadas de instrumentación en ubicaciones arbitrarias del código fuente de la aplicación objetivo. Estas llamadas registran todas las peticiones de memoria de la aplicación en una traza de programa. La distribución de Pin incluye muchos ejemplos de herramientas independientes de la arquitectura como analizadores, simuladores de cache, revisores de traza y verificadores de errores de memoria. Una parte del código que permite extraer las direcciones y el tamaño de cada asignación y liberación de memoria se muestra en la Figura 6.4

```
.....
.....

VOID MallocBefore(CHAR * name, ADDRINT addr){
 TraceFile << name << "u" << (1.0*clock())/CLOCK_PER_SEC << "u" <<size;
}

VOID FreeBefore(CHAR * name, ADDRINT addr){
 TraceFile << name << "u" << (1.0*clock())/CLOCK_PER_SEC << "u" <<size << endl;
}

VOID Mallocafter(ADDRINT addr){
 TraceFile << "u" << addr << endl;
}

VOID Image(IMG img, VOID *v){
 RTN mallocRtn = RTN_FindByName(img, MALLOC);
 if (RTN_Valid(mallocRtn) {
  RTN_Open(mallocRtn);
  RTN_InsertCall(mallocRtn, IPOINT_BEFORE, (AFUNPTR)MallocBefore, IARG_ADDRINT, MALLOC,
       IARG_G_ARG0_CALLEE, IARG_END);
  RTN_InsertCall(mallocRtn, IPOINT_AFTER, (AFUNPTR)MallocAfter, IARG_ADDRINT, MALLOC,
       IARG_G_RESULT0, IARG_END);
  RTN_Close(mallocRtn);
  }

  RTN freeRtn = RTN_FindByName(img, FREE);
 if (RTN_Valid(freeRtn) {
  RTN_Open(freecRtn);
  RTN_InsertCall(freeRtn, IPOINT_BEFORE, (AFUNPTR)FreeBefore, IARG_ADDRINT, FREE,
       IARG_G_ARG0_CALLEE, IARG_END);
  RTN_Close(freeRtn);
  }
 }
 ......
 ......

 int main (int argc, char *argv[]) {
  PIN_InitSymbols();
  if (PIN_Init(argc,argv))
    return Usage();

  TraceFile.open(KnobOutputFile.Value().c_str());

  IMG_AddInstrumentFunction(Image, 0);
  PIN_AddFineFunction(Fini, 0);

  PIN_StartProgram();

  return 0;
}
```

Figura 6.4: Porción de código utilizado en la instrumentación con la herramienta PIN para el seguimiento de las llamadas *malloc()* y *free()* durante la ejecución de las aplicaciones.

```
 make dir DmmProfile.so.test
../../../pin -t obj -intel32 /DmmProfile.so -- <app>
```

Figura 6.5: Perfilado de la aplicación.

Como resultado, después de ejecutar la aplicación, la traza de programa está disponible y el diseñador del sistema puede evaluar diferentes DMM utilizando la mis-

ma traza de la aplicación. De esta forma, la aplicación sólo debe ser ejecutada una vez. La Figura 6.5 muestra cómo se ejecuta el perfilado de la aplicación. La primera línea de la Figura 6.6 compila el código de instrumentación, después de incluir el fichero DmmProfile.cpp en el *makefile* correspondiente. La segunda línea ejecuta la aplicación objetivo original, especificada en el parámetro <*app*> que es el binario original de la aplicación objetivo. Es interesante mencionar que utilizando esta herramienta de instrumentación, la aplicación original no necesita ser modificada para el uso de librerías dinámicas.

```
ProfilingReport profReport = new ProfilingReport() ;
profReport.load("profile.mem" ) ;
ExactSegregatedFit exact = new ExactSegregatedFit(0 , profReport.getMaxSizeInB(), profReport.
    getSizesInB()) ;
exact.setup(FreeList.DATA_STRUCTURE.SLL, FreeList.ALLOCATION_MECHANISM.FIRST , FreeList.
    ALLOCATION_POLICY.FIFO ) ;

DynamicMemoryManager manager = new DynamicMemoryManager(exact);
Simulator simulator = new Simulator(profReport, manager);

simulator.start();
simulator.join();
```

Figura 6.6: Configuración de un DMM utilizando la API del simulador.

La Figura 6.6 muestra como se aplica la librería para crear un DMM compuesto, después de la generación de las trazas de la aplicación. En el ejemplo, en primer lugar se lee la traza de la aplicación (líneas 1-2), generadas anteriormente con Pin. Cuando se construye el gestor *ExactSegregatedFit*, se proporciona el constructor con los tamaños mínimo y máximo de bloque en bytes y los diferentes tamaños de bloque soportados por este gestor. En este ejemplo de la Figura 6.6, línea 3, se proporcionan los dos últimos parámetros mediante la traza de la aplicación, que también pueden ser establecidos manualmente. A continuación, se configura el gestor de memoria definiendo la estructura de datos a utilizar (lista simplemente enlazada), el mecanismo de asignación (*first-fit* y la política de asignación (*first in*, *first out*). A continuación, se construye el DMM diseñado. Tal y como se ha mencionado previamente, un DMM puede contener uno o más gestores en el caso que se presenta. Por último, se llama al simulador (líneas 8-10), en la Figura 6.6 y después de pocos

segundos se obtienen la métricas necesarias para evaluar el DMM.

Al mismo tiempo que se ejecuta la simulación, se calculan varias métricas importantes, como el número de asignaciones/liberaciones, *splittings*, *coalescings*, rendimiento, uso de memoria y accesos a memoria. Todos estos parámetros excepto el tiempo de ejecución se pueden calcular con exactitud. Sin embargo, el sistema está utilizando tiempo de simulación en lugar de tiempo real, el tiempo total de ejecución se calcula como la complejidad computacional o complejidad de tiempo [200].

```
void firstFit (long sizeInB) {
// ...
while (iterator.hasNext()) {
counterForMetrics++;
currentBlock = iterator.next() ;
if (currentBlock.sizeInB >= sizeInB) {
block = currentBlock;
iterator.remove();
break ;
}
}
metrics.addExecutionTime(counterForMetrics);
metrics.addMemoryAccesses(2*counterForMetrics);
// ...
}
```

Figura 6.7: Cálculo del tiempo de ejecución y accesos a memoria.

El fragmento de código en la Figura 6.7 ilustra un ejemplo de cómo se realiza esta tarea en el marco de optimización de DMM propuesto. Este extracto de código muestra una parte del algoritmo *first-fit* dentro del simulador. El bucle principal busca el primer bloque suficientemente grande para ubicar el tamaño solicitado. De esta forma, se cuenta el número de iteraciones en el bucle y después, el tiempo de ejecución y los accesos a memoria se modifican como corresponde +1 por cada ciclo en el bucle para considerar el tiempo computacional y +2 por cada ciclo para contar dos accesos en el gestor actual: uno para el nodo actual en la lista de bloques libres y otro para calcular el tamaño, por ejemplo, resta de dos punteros).

En un estudio preliminar, se analizó el impacto que los DMM tienen en las aplicaciones y en el dispositivo de memoria utilizando el simulador DMM previamente descrito. Con ese objetivo se emplearon métricas y aplicaciones comúnmente uti-

lizadas para evaluar el rendimiento y la fragmentación de los gestores de memoria dinámica [93] y descritas en [201, 202]. Concretamente, se analizaron cuatro métricas: tiempo de ejecución, uso de memoria, temperatura del dispositivo de memoria (modelo térmico propuesto en [50]) y consumo de energía (modelo de energía de memoria desarrollado en [203]). Las pruebas se realizaron con cinco aplicaciones intensivas de memoria: *Lindsay*, *Cfrac*, *GCbench*, *Expresso* y *Roboop*. La Tabla 6.1 muestra los resultados obtenidos en promedio tomando el DMM Lea, como DMM implementado en el sistema GNU/Linux. El DMM obtenido presenta una influencia realmente importante en cuanto al rendimiento y uso de la memoria que alcanza el $43,25\,\%$ y $22,9\,\%$, respectivamente, siendo poco significativa respecto al incremento de temperatura y el consumo de energía, que se sitúa en promedio en $0,0006\,\%$ y $0,48\,\%$.

Tabla 6.1: Estudio preliminar: resultados en porcentaje promedio de incremento de tiempo de ejecución, uso de memoria, temperatura y consumo de energía.

Tiempo de Ejecución	Uso Memoria	Inc. Temperatura (K)	Energía (mW)
43,25 %	22,90 %	0,0006 %	0,48 %

De acuerdo con los resultados anteriormente mencionados, se plantea un marco de optimización basado en GE para el diseño de gestores de memoria dinámica a medida de las aplicaciones objetivo. Rendimiento y uso de memoria se utilizan como métricas para evaluar la idoneidad de cada DMM y se actualiza el conjunto de aplicaciones a un subconjunto de las aplicaciones SPEC, ampliamente utilizadas. A continuación se describe el marco de optimización desarrollado.

6.3. Optimización de DMM mediante Gramáticas Evolutivas

En esta sección se describe el marco de optimización basado en Gramáticas Evolutivas (GE) (metaheurística ya abordada en el Capítulo 3 y utilizada en la sección 5.4 en el Capítulo 5) y perfilado estático de las aplicaciones. Este método permite evaluar de forma automática implementaciones complejas de gestores de

memoria dinámica para aplicaciones que presentan una carga alta de operaciones de memoria dinámica, como es el caso de las aplicaciones multimedia, sin necesidad de integrarlas en la aplicación. La metodología propuesta, mostrada en la Figura 6.1, se divide en tres fases: (i) obtener las trazas de las aplicaciones objetivo, para lo cual se emplea la herramienta de instrumentación Pin; (ii) analizar la traza de la aplicación que permite la definición de la gramática que mejor se adapta a los patrones de cada aplicación; (iii) ejecutar el algoritmo de optimización basado en GE (GEA), que implica la ejecución del simulador DMM (ver sección 6.2), con el que se calcula el valor objetivo de cada DMM a evaluar.

6.3.1. Definición de la gramática BNF

Para abordar el objetivo de diseñar DMM optimizados a una aplicación objetivo mediante GEA, es necesario definir la gramática en formato BNF. Con este propósito, la gramática sigue la clasificación de los DMM que se describe en la sección 6.2. La definición de la gramática depende de los tamaños de bloque que manipula cada aplicación, por esta razón se ha generado un fichero que contiene la gramática a partir de la traza de la aplicación objetivo. En la Figura 6.8 se muestra la gramática generada para la aplicación DealII, una de las aplicaciones analizadas. Esta gramática está formada por 12 reglas de producción, 12 símbolos no terminales y 31 terminales. De acuerdo a las reglas de producción un DMM es una lista de varios gestores (reglas I y II). Los dos tipos de gestores que hay, según la regla II, son: *AllocatorSize*, que puede preceder a más gestores, y *AllocatorMaxSize* que es el último gestor en la cadena. De igual forma, las reglas de producción III y IV son equivalentes con la única diferencia de que un *AllocatorMaxSize* se usa para gestionar los tamaños de bloque pertenecientes al rango más alto. El comportamiento general del gestor de memoria se define en su *AllocatorClass* que especifica la forma en que se distribuyen las listas (o matrices) de bloques. En este enfoque se han implementado hasta 5 clases diferentes que se especifican en la regla V. Las dos reglas de producción *AllocatorSize* o *AllocatorMaxSize* determinan si el gestor permite dividir un bloque

en varios (*splitting*) y/o unir varios bloques en uno (*coalescing*). Cada gestor es responsable de gestionar los tamaños de bloque en el rango comprendido entre (*Size bytes* y *Size*] definido en el siguiente gestor. Si el gestor, por ejemplo, es el último y es *AllocatorMaxSize*, entonces gestiona los tamaños de bloques en el rango (*Size*, *MaxSize*]. Las reglas de producción *Size* y *MaxSize* (VI y VII) dependen de la traza de la aplicación objetivo, debido a que cada aplicación trabaja con su propio rango de tamaños de bloque.

Respecto al tamaño de bloques utilizado por las aplicaciones, se ha observado que diferentes trazas de la misma aplicación presentan mucha similitud tanto en tamaños de bloque como en el número de *malloc* y *free*. Esto sucede incluso en aplicaciones puramente interactivas como es el caso de muchas aplicaciones multimedia (por ejemplo los videojuegos que necesitan de la interacción humana). Como consecuencia de ello, sólo se necesita una traza de la aplicación para obtener un DMM optimizado.

Además de los tamaños de bloque, también se declaran en la regla X las estructuras que el gestor utiliza para almacenar los bloques (se han declarado tres estructuras de datos), los mecanismos de asignación se encuentran en la regla XI (*first*, *best* y *exact fit*)[1] y la política de asignación en la regla XII (*FIFO* y *LIFO*)[2].

Cada individuo de GE utiliza un esquema de codificación de longitud variable, donde cada gen almacena un valor entero que se mapea a una de las reglas de producción, anteriormente etiquetadas en un formato BNF mediante el proceso de decodificación.

La Figura 6.9 representa un ejemplo de genoma que tiene 18 genes con valores enteros que se decodifica con la gramática de la Figura 6.8. El proceso de decodificación aplica la operación módulo entre el valor del gen y el número de opciones de cada regla de producción. De esta forma, el valor máximo para cada gen es el

[1] **first**: primer ajuste que asigna el primer bloque cuyo tamaño satisfaga la petición; **best**: mejor ajuste y asigna el bloque con el tamaño más pequeño que satisfaga la petición; **exact fit**: ajuste óptimo que asigna el bloque cuyo tamaño sea igual al solicitado.

[2] FIFO (First Input First Output): una inserción en la estructura se realiza siempre por la cola; LIFO (*Last Input First Output*), una inserción se realiza siempre por la cabeza de la estructura.

```
N = {<DynamicMemoryManager>,<Allocators>,<AllocatorSize>,<AllocatorMaxSize>,<AllocatorClass>,<
     Size>,<MaxSize>,
     <AllowSplitting>,<AllowCoalescing>,<DataStructure>,<AllocationMechanism>,<
          AllocationPolicy>}
T = {SegregatedFreeList,SimpleSegregatedStorage,ExactSegregatedFit,BuddySystemBinary,
     BuddySystemFibonacci,4,7,8,
     (omitidos para simplificar)...,7832240,true,false,SLL,DLL,BTREE,FIRST,BEST,EXACT,FIFO,
          LIFO}
S = <DynamicMemoryManager>
I <DynamicMemoryManager>::=<Allocators>
II <Allocators> ::=<AllocatorSize>
              |<AllocatorMaxSize>
III <AllocatorSize>::=<AllocatorClass>
                     <AllowSplitting>
                     <AllowCoalescing>
                     <Size>
                     <DataStructure>
                     <AllocationMechanism>
                     <AllocationPolicy>
                     <Allocators>
IV <AllocatorMaxSize> ::=<AllocatorClass>
                     <AllowSplitting>
                     <AllowCoalescing>
                     <MaxSize>
                     <DataStructure>
                     <AllocationMechanism>
                     <AllocationPolicy>
V <AllocatorClass> ::=SegregatedFreeList
                     |SimpleSegregatedStorage
                     |ExactSegregatedFit
                     |BuddySystemBinary
                     |BuddySystemFibonacci
VI <Size
     >::=4|7|8|12|14|15|16|17|18|19|20|21|23|24|25|26|28|31|32|36|40|44|48|52|56|60|64|72|76|79
          |80|84|88|92|96|100|104|108|112|116|120|128|132|136|140|144|152|160|168|172|176|180|184

          |188|192|200|202|208|212|216|220|224|228|232|236|240|248|252|256|264|276|280|284|288|300

          |303|312|316|320|324|336|348|352|356|360|380|384|396|400|424|432|440|444|448|457|480|492

          |496|500|504|512|525|528|536|540|556|564|576|584|600|608|617|624|626|630|631|632|648|660

          |672|692|713|720|745|756|768|772|784|800|808|820|...|820840|995732|1011432|1034632|1274436

          |1295112|1296000|1616064|1916188|1947512|1991464|2548872|2760556|2804912|3832376|3916120

          |3979304|5521112
VII <MaxSize> ::=7832240
VIII <AllowSplitting> ::=true|false
IX <AllowCoalescing> ::=true|false
X <DataStructure> ::=SLL|DLL|BTREE
XI <AllocationMechanism> ::=FIRST|BEST|EXACT
XII<AllocationPolicy> ::=FIFO|LIFO
```

Figura 6.8: Fichero con la gramática generada para la aplicación DealII, a partir de su traza. Se han omitido más de 200 tamaños de bloque por simplicidad. Las diferencias entre los ficheros con las gramáticas de las aplicaciones abordadas, se dan en los símbolos no terminales *Size* y *MaxSize*.

401	213	8	151	77	2	34	60	300	114	205	7	2	122	183	197	49	136

Figura 6.9: Ejemplo de genoma de un individuo de GE.

número máximo de opciones de una regla en la gramática dada.

El proceso de decodificación comienza con la lectura del primer gen, 401. Dado que el grupo identificado como **I** sólo tiene una regla producción, concretamente *DynamicMemoryManager*, se selecciona la regla de producción 0 (401 mod 1=0), que corresponde a <*DynamicMemoryManager*>*::=*<*Allocators*>. El genotipo se usa para mapear el símbolo inicial, tal y como se define en la gramática, a símbolos terminales mediante la lectura de codones para generar el valor entero que le corresponde. A continuación, se lee el segundo gen cuyo valor es 213 y después de realizar la operación módulo (213 mod 2), el resultado es 1; así, se selecciona la regla de producción <*Allocators*> *::=* <*AllocatorSize*>. <*AllocatorSize*> tiene 8 producciones que han de ser decodificadas y se seleccionan los genes 8, 151, 77, 2, 34, 60, 300 y 114. Una vez realizada la operación módulo, el resultado de la decodificación es:

- AllocatorClass (8 mod 5 = 3): BuddySystemBinary
- AllowSplitting (151 mod 2 = 1): false
- AllowCoalescing (77 mod 2 = 1): false
- Size (2 mod 419 = 2): 8
- DataStructure (34 mod 3 = 1): DLL
- AllocationMechanism (60 mod 3 = 0): FIRST
- AllocationPolicy (300 mod 2 = 0): FIFO
- Allocators (114 mod 2 = 0): AllocatorMaxSize

En este punto, la expresión decodificada contiene un gestor que utiliza un comportamiento *BuddySystemBinary*. No permite *splitting* ni *coalescing*. Además, el DMM decodificado hasta este punto maneja tamaños de bloque en el rango de (0,8]. Las estructuras de datos empleadas para almacenar bloques es una lista doblemente enlazada. Por último, el mecanismo de asignación y la política de asignación son *first* y *FIFO*, respectivamente. Se sigue el mismo procedimiento hasta completar el fenotipo del individuo y se decodifican los siete genes siguientes (205, 7, 2, 122, 183, 197 y 49), que después de aplicar la operación módulo, se obtiene como resultado:

- AllocatorClass (205 mod 5 = 0): SegregatedFreeList
- AllowSplitting (7 mod 2 = 1): false
- AllowCoalescing (2 mod 2 = 0): true
- MaxSize (122 mod 1 = 0): 7832240
- DataStructure (183 mod 3 = 0): SLL
- AllocationMechanism (197 mod 3 = 0): FIRST
- AllocationPolicy (49 mod 2 = 1): LIFO

Tabla 6.2: Gestor de memoria dinámica resultado de decodificar el genoma de la Figura 6.9.

Estructura de datos	Mecanismo(Política)	Rango (bytes)
BuddySystemBinary, split=false, coalesce=false		
DLL	FIRST(FIFO)	(0,1]
DLL	FIRST(FIFO)	(1,2]
DLL	FIRST(FIFO)	(2,4]
DLL	FIRST(FIFO)	(4,8]
SegregatedFreeList, split=false,coalesce=true		
SLL	FIRST(LIFO)	(8,7832240]

Después del gen 17 se tiene el DMM decodificado que se representa en la Tabla 6.2. El DMM está compuesto por dos gestores, ambos definidos de acuerdo al individuo descrito en la Figura 6.9. De esta forma, durante el proceso de optimización en la Figura 6.1(c), el simulador recibe el mejor individuo, se decodifica y se simula el DMM resultante utilizando la traza de la aplicación y calculando su valor objetivo en términos de rendimiento y uso de memoria.

Una ventaja importante de la representación de las gramáticas evolutivas, tal y como se ha mencionado en capítulos anteriores, es la utilización de un genoma linear. Esto permite que GE pueda usar directamente los operadores genéticos estándar.

De hecho, en el ejemplo anterior el proceso de decodificación finalizó sin traducir todos los genes. Esto sucede porque en GE las operaciones genéticas no conocen la semántica del genoma hasta que no se decodifica, por tanto, el proceso de decodificación frecuentemente finaliza con una expresión completa sin recorrer el genoma

completo. Por esta razón, el proceso de exploración se realiza en un periodo de tiempo corto.

Por otra parte, es posible encontrar un problema más serio con GE, que el hecho de existir genes redundantes. En el proceso de decodificación podría suceder que un genoma no tuviera genes suficientes para crear un genotipo completo, quedando aún símbolos no terminales en el cromosoma. En este caso, se aplica el operador *wrap* que consiste en comenzar a leer desde el primer codón en el cromosoma. Incluso después de aplicar la primera vez el operador *wrap*, el proceso de mapeo podría no completarse y continuar así indefinidamente. Esto sucede porque los símbolos no terminales se mapean recursivamente mediante varias reglas de producción. En este caso, el individuo sería marcado como inválido. Por esta razón, en este enfoque de optimización para DMM se ha establecido un límite máximo en el número de veces que se puede aplicar el operador *wrap* (*wrappings*). Así, durante el proceso de mapeo que comienza desde la parte izquierda del genoma, se generan valores enteros que se utilizan para seleccionar reglas de la gramática BNF, hasta que se alcanza una de las situaciones siguientes:

1. Se genera el DMM completo. Ello sucede cuando todos los símbolos no terminales se convierten en elementos del conjunto de terminales de la gramática BNF.

2. Se alcanza el final del genoma y se llama al operador *wrap* que comienza leyendo el genoma desde el extremo izquierdo otra vez. La lectura de codones continúa, a menos que se alcance el número máximo de *wrappings*, durante la decodificación del individuo.

3. Si se alcanza el número máximo de *wrappings* y el individuo aun no se ha mapeado completamente, se para el proceso de mapeo y se asigna al individuo el valor objetivo más alto (asumiendo minimización).

La gramática es lo suficientemente completa como para implementar cualquier DMM conocido y explorar implementaciones personalizadas de DMM para aplicaciones multimedia reales, tal y como se muestra en la sección 6.3.3. En cuanto a las plataformas hardware, puede ser aplicada a cualquier plataforma que soporte C++.

6.3.2. Configuración de los experimentos

En esta sección, se describe el conjunto de aplicaciones utilizadas, la función objetivo empleada y la configuración general del algoritmo de optimización.

Aplicaciones

Como se ha mencionado anteriormente, en esta tesis se estudian las implicaciones en el rendimiento y el uso de memoria cuando se construyen gestores de memoria de propósito general y a medida de una aplicación utilizando el simulador DMM y el algoritmo propuesto GEA, basado en GE. Con el fin de evaluar ambas métricas, se ha seleccionado un número de aplicaciones intensivas de memoria pertenecientes al conjunto de aplicaciones SPEC, que son descritas brevemente a continuación, algunas de las cuales se han ejecutado con diferentes entradas, tal y como se indica en la Tabla 6.3. Estas aplicaciones se han actualizado respecto a los experimentos realizados en el estudio preliminar, mencionado anteriormente, por ser más actuales, incrementar la presión sobre el subsistema de memoria, entre otros, y ser ampliamente utilizados por la comunidad científica.

Tabla 6.3: Estadísticas de las aplicaciones intensivas de memoria analizadas.

Aplicación	Objetos	Memoria total (bytes)	Max en uso (bytes)	Tamaño promedio (bytes)	Op. memoria
hmmer	5000101	2780155771	66074	556,01	10020365
dealII	4926536	451183299	33747256	91,58	10251022
soplex	6188584	4537088927228	13273944	733138,45	12377148
calculix	4054631	19675406468	92335483	485257	10432894
gcc/200	4810046	4568294683	163997226	949,74	10875348
gcc/expr2	3363661	6246880022	531906484	1857,16	7632761
gcc/c-typeck	2786057	6040320719	408900497	2168,05	6781339
perl/split	5534308	1526190192	140256527	275,76	10900336
perl/diff	5321169	54544092	17209415	10,25	10544463
perl/spam	5278732	213309132	27784844	40,40	10320179

hmmer realiza búsquedas sensibles en una base de datos de secuencias de genes, aplicando los modelos estadísticos desarrollados en los perfiles de modelos ocultos de Markov. Estas técnicas son ampliamente utilizadas en computación biológica para buscar patrones en secuencias de ADN y en el análisis de secuencias de proteínas.

dealII es una aplicación que utiliza la librería de C++ denominada deal.II. Esta

librería permite el desarrollo de algoritmos adaptativos de elementos finitos en varias dimensiones espaciales, entre otros aspectos, estimaciones sofisticadas de error y redes adaptables. La librería proporciona una moderna interface para las estructuras de datos complejas y los algoritmos necesarios. Los programas también pueden ser desarrollados para ser independientes de la dimensión espacial, sin demasiada penalización en cuanto al tiempo de ejecución y consumo de memoria.

soplex es una aplicación que resuelve un problema de programación lineal aplicando el algoritmo *Simplex*. Un problema de programación lineal está definido por una matriz dispersa de $m \times n$, y a la derecha del vector b de m dimensiones y un vector c de coeficientes de la función objetivo de n dimensiones. Generalmente, el objetivo es encontrar el vector x que minimiza c con $Ax \leq b$ y $x \geq 0$.

calculix se aplica en mecánica estructural y se basa en *Calculix*, software que utiliza la teoría clásica de elementos finitos. Se emplea para aplicaciones estructurales de tres dimensiones no lineales y puede ser utilizado para resolver problemas relativos al diseño de puentes, edificios, resistencia a terremotos, fenómenos de resonancia, etc.

gcc es un compilador optimizado del lenguaje C basado en la versión de *gcc 3.2*. Añade generación de código optimizado para un procesador AMD. Las heurísticas en línea se han modificado ligeramente para añadir un código más típico del uso actual del compilador. Además, gcc podría consumir más tiempo analizando las entradas de código fuente y utilizar más memoria. Esta aplicación se ha perfilado con tres entradas distintas: 200.i que viene de una versión anterior de las aplicaciones SPEC2000 de punto flotante (SPECfp2000), *200.sixtrack* (*gcc/200*). *Expr2.i* procede del programa fuente *403.gcc* (*gcc/expr2*), al igual que *c-typeck.i* de (*gcc/c-typeck*).

perlbench es una versión reducida de PERL, un lenguaje de programación de *script* donde se han eliminado las características específicas del sistema operativo. La carga de trabajo consiste en tres *scripts* con varios módulos de terceros. La primera se corresponde con el software de código abierto *SpamAssasin* que com-

prueba *spam*, y se usa para marcar varios corpus conocidos tanto de *spam* como *ham*. *MhonArc* es un conversor libre de formato correo a html, donde los mensajes de correo se generan aleatoriamente y se convierten a html. Por último, una versión ligeramente modificada del *script specdiff*, como parte del conjunto de herramientas CPU2006. De esta forma, el perfilado de esta aplicación se ha realizado con 3 entradas diferentes *splitmail*, basada en el programa *splitmail* que toma un mensaje de correo y lo trocea en partes pequeñas (perl/split); *diffmail*, que es una arquitectura de entrega de mensajes diferenciados para el control de correo no deseado (Perl/-diff); y *checkspam*, que es un script en Perl para comprobar tanto *spam* como *ham* usando el módulo *SpamAssassin* (Perl/spam).

El perfilado de las aplicaciones se realiza con la herramienta Pin, tal y como se ha descrito en la sección 6.2. Todas las aplicaciones se han ejecutado en un sistema Q8300 con procesador Intel Core 2 Quad con 4 GB de RAM, con Windows 7. Esta tarea sólo se realiza una vez en la metodología de exploración de DMM propuesta.

La Tabla 6.3 incluye el número de objetos asignados y su tamaño promedio. El rango del uso de memoria de las aplicaciones va desde los $64,5KB$ (para *hmmer*), a más de $507,27MB$ (para gcc/expr2). Para todos los programas, la relación entre la memoria total asignada y la cantidad máxima de memoria en uso es grande. Además, el número de operaciones de memoria también es grande. Por lo tanto, todas las aplicaciones propuestas tienen una dependencia alta del subsistema de memoria dinámica. En el siguiente paso se simulan cinco gestores de memoria de propósito general y se realiza una exploración automática de DMM viables usando gramáticas evolutivas sobre todas las aplicaciones objetivo.

Función objetivo

Con el fin de seleccionar el mejor DMM posible entre un conjunto de candidatos, el algoritmo de optimización debe usar una función objetivo bien definida. Así, la evaluación de cada DMM se realiza con la función objetivo representada en la Ecuación (6.1), donde el tiempo de ejecución y el uso de la memoria tienen

asignado el mismo peso y están normalizados a los correspondientes de los DMM Kingsley y Lea, considerados como el más rápido y el más eficiente de los DMM, respectivamente. Así, T y M son el tiempo de ejecución y el uso de la memoria para el DMM que se está evaluando y T_{Kng} y M_{Lea} son el tiempo de ejecución y el uso de la memoria para los DMM Kingsley y Lea, respectivamente. Es importante destacar que se define el uso de memoria como la cota máxima de memoria solicitada a la memoria virtual.

$$F = 0{,}5 \times \frac{T}{T_{Kng}} + 0{,}5 \times \frac{M}{M_{Lea}} \tag{6.1}$$

Configuración del algoritmo

El algoritmo GE se ha implementado utilizando la librería de optimización desarrollada en Java JECO. Los parámetros de configuración del algoritmo se muestran en la Tabla 6.4. Se han seleccionado 250 generaciones y 100 individuos porque es el número mínimo de evaluaciones necesarias mediante GEA para alcanzar el mejor DMM en la aplicación *Soplex*, que es el más grande en términos de operaciones de memoria (12×10^6, como se muestra en la Tabla 6.3). Evidentemente, con el número fijo evaluaciones, cuanto menor es el tamaño de la traza (respecto a *Soplex*), mayor calidad en las soluciones.

Tabla 6.4: Parámetros de configuración del algoritmo GE.

Parámetro	Valor
Tamaño de población	100
Tamaño de elitismo	10
Tipo de reemplazo	generacional
Número de generaciones	250
Mecanismo de selección	Torneo
Tamaño de torneo	2
Probabilidad de cruce	0,80
Punto fijo de cruce	yes
Probabilidad de mutación	0,02
Máximo número de *wraps*	3

Con todo ello, hay que destacar que en la metodología propuesta el simulador puede obtener el valor objetivo de un DMM genérico en un par de segundos (el DMM Kingsley, por ejemplo), y continúa con el proceso de optimización hasta que

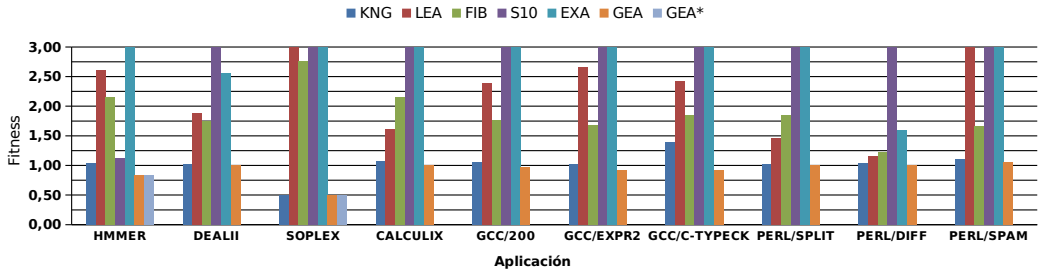
se alcanza un valor objetivo dado (por ejemplo, el 75 % del valor objetivo obtenido por Kingsley).

6.3.3. Resultados

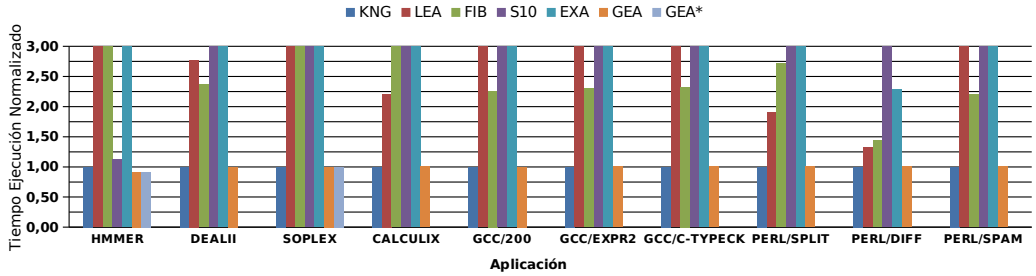
En las pruebas realizadas se han simulado las aplicaciones descritas en la sección anterior, considerando cinco gestores de memoria de propósito general: Kingsley (KNG), Doug Lea (LEA), *buddy system* basado en el algoritmo de asignación Fibonacci (FIB), una lista de *10 segregated free-lists* (S10) y un gestor *exact segregated free list* (EXA). Los resultados se comparan con los DMM personalizados obtenidos con el proceso de exploración automática GEA. Para encontrar una solución al problema de optimización multi-objetivo se construye una única función objetivo global mostrada en la Ecuación (6.1). Después de analizar los resultados obtenidos con la configuración global, las aplicaciones *hmmer* y *soplex* no obtienen mejora. Por ello, para comprobar que el algoritmo no cae en un óptimo local, se incrementa el número de generaciones para las aplicaciones *hmmer* y *soplex* a 5000 (etiquetado como GEA$*$). Los resultados obtenidos son el promedio de las 10 ejecuciones lanzadas.

Para crear los DMM a medida, se han seguido las fases metodológicas presentadas en la Figura 6.1. En primer lugar se ha obtenido la traza de la aplicación mediante la herramienta Pin. Esta traza sirve de entrada al generador de gramáticas que recoge los diferentes tamaños de bloque solicitados por la aplicación, dato desconocido hasta ese momento, y proporciona el fichero con la gramática. La última fase es la responsable de ejecutar el algoritmo con gramáticas evolutivas.

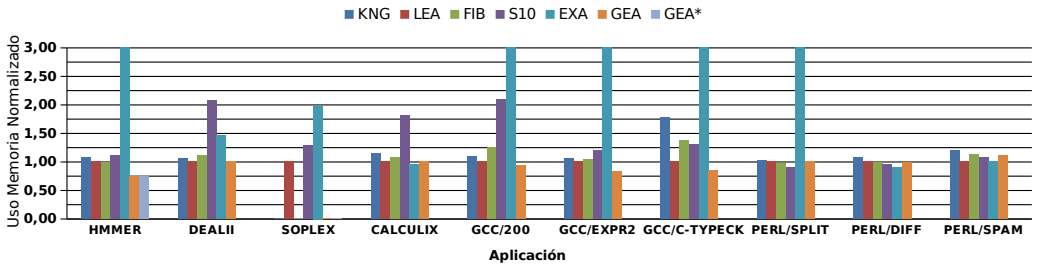
Los resultados experimentales se presentan en la Figura 6.10. El límite superior de la escala de valores en el diagrama de barras se ha establecido a 3. Sin embargo, el valor objetivo global en *soplex* para EXA es muy superior (F_{EXA} = 2763,79). Esto se debe a que el tiempo de ejecución es 4 órdenes de magnitud mayor que con Kingsley ($2,05\times10^{11}$ frente a $3,71\times10^{7}$). Lo mismo sucede con S10 para *dealII*, *gcc/expr2*, *gcc/c-typeck* y *perl/split* que alcanzan valores en tiempo de ejecución cuatro



(a) Valor objetivo global



(b) Tiempo de Ejecución



(c) Uso de memoria

Figura 6.10: Valor objetivo global F, tiempo de ejecución (T/T_{KNG}) y uso de memoria (M/M_{LEA}) para las seis aplicaciones gestionadas por seis DMMs. En promedio, GEA se presenta como la mejor opción como se puede comprobar en los resultados que se muestran en la tabla 6.6.

órdenes de magnitud mayor que Kingsley para *dealII* y tres órdenes de magnitud para las tres aplicaciones mencionadas. En cuanto al uso de la memoria, el único DMM que se sale de esta cota es EXA para cinco de las aplicaciones abordadas. La Figura 6.10.(a) representa el valor objetivo global alcanzado por todos los DMM propuestos, aplicando la Ecuación (6.1). La Figura 6.10.(b) y (c) presenta los componentes de la valor objetivo global, de forma independiente, es decir, el tiempo de ejecución normalizado (T/T_{KNG}) y el uso de memoria (M/M_{LEA}), respectivamente. En todos los experimentos realizados, la desviación típica es inferior a 0,04.

En la Tabla 6.5 se presenta la mejora promedio que obtiene GEA en cuanto al

Tabla 6.5: Porcentaje de mejora promedio. GEA vs. KNG, LEA, FIB, S10 y EXA.

	KNG	LEA	FIB	S10	EXA	Average
Valor Objetivo = $100 \times \frac{F_* - F_{GEA}}{F_*}$	9,13 %	62,52 %	51,81 %	86,88 %	86 %	59,27 %
Rendimiento = $100 \times \frac{T_* - T_{GEA}}{T_*}$	1,17 %	72,44 %	62,62 %	85,74 %	90,78 %	62,55 %
Memoria = $100 \times \frac{M_* - M_{GEA}}{M_*}$	16,03 %	23,14 %	15,08 %	38,88 %	59,96 %	30,62 %

valor objetivo global, rendimiento y uso de memoria, respectivamente, para las seis aplicaciones. Se puede observar que GEA supera a los cinco DMM en todos los casos.

Como análisis global de los DMM de propósito general, se puede ver en la Figura 6.10(a) y en la Tabla 6.5 que el DMM Kingsley es un gestor de memoria excelente. Kingsley es claramente el mejor DMM de propósito general en términos de rendimiento (ver Figura 6.10(b)). Sin embargo, este gran rendimiento se ve ligeramente perjudicado por el uso de memoria (ver Figura 6.10(c)) donde Kingsley es superado, principalmente por LEA, en nueve de los 10 casos. Como consecuencia, LEA es un firme candidato para reducir el uso de memoria. Es importante destacar que LEA es ampliamente superado por Kingsley en la aplicación *soplex*. *Buddy system* basado en Fibonacci (FIB) presenta siempre un comportamiento intermedio en cinco de las seis aplicaciones. Los DMM de propósito general S10 y EXA tienen un comportamiento diferente dependiendo de la aplicación. Por ejemplo, examinando el valor objetivo global Figura 6.10(a), S10 es un buen gestor para *hmmer*, pero es la peor opción posible para *dealII*, *calculix*, *gcc* y *perlbench*. Por último, la lista con ajuste exacto diferenciado (EXA) no han dado una buena respuesta, siendo la peor opción para *hmmer* y *soplex*.

Tabla 6.6: Porcentaje de mejora GEA vs. Kingsley.

	$100 \times \frac{F_{KNG}-F_{GEA}}{F_{KNG}}$	$100 \times \frac{T_{KNG}-T_{GEA}}{T_{KNG}}$	$100 \times \frac{M_{KNG}-M_{GEA}}{M_{KNG}}$
hmmer	20,08 %	8,33 %	31,4 %
hmmer*	20,08 %	8,33 %	31,4 %
dealII	2,31 %	0,11 %	4,58 %
soplex	0,17 %	0,08 %	3,72 %
soplex*	0,17 %	0,08 %	3,72 %
calculix	6,59 %	-0,84 %	13,17 %
gcc/200	7,21 %	0,01 %	14,05 %
gcc/expr2	10,55 %	-0,37 %	20,80 %
gcc/c-typeck	33,35 %	-0,49 %	52,44 %
perl/split	1,11 %	-0,04 %	2,05 %
perl/diff	3,61 %	-0,58 %	7,65 %
perl/spam	4,30 %	-0,54 %	8,14 %

A continuación se analizan los resultados obtenidos por GEA y GEA*, examinando cada métrica independientemente. A tal efecto, en la Tabla 6.6 se presenta el porcentaje de mejora alcanzado por el algoritmo de exploración, comparado con el gestor de memoria Kingsley. Desde el punto de vista del valor objetivo global (ver Figura 6.10(a)), se concluye que GEA es la mejor opción en las seis aplicaciones y 10 configuraciones distintas, proporcionando mejoras del 20,08 %, 2,31 %, 0,17 %, 6,59 %, 7,21 %, 10,55 %, 33,35 %, 1,11 %, 3,61 % y 4,30 % para *hmmer*, *dealII*, *soplex*, *calculix*, *gcc* (200, *epxr2*, *c-typeck*), y *perlbench* (*split*, *diff*, *spam*), respectivamente. Es de destacar que los resultados para 5000 generaciones han sido exactamente los mismos que para 250, como se puede observar para *hmmer** y *soplex** en la Tabla6.6. Es lógico, debido a que tanto GEA como GEA* finalizan con valores muy cercanos al rendimiento de Kingsley, lo cual es difícil de mejorar.

En cuanto al rendimiento (Figura 6.10(b)), Kingsley y GEA obtienen el mejor resultado. Observando los resultados de ambos, en 6 de los 10 casos GEA es peor que Kingsley, pero el porcentaje de empeoramiento es únicamente, algo más del $0,84$ %. Este dato no es demasiado relevante, dado que Kingsley es un gestor que optimiza en gran medida el rendimiento, mientras que trabaja peor que otros gesto-

res con la memoria [197]. Por tanto, LEA, FIB, S10 y EXA, por el contrario, tienen un comportamiento diferente que depende de la aplicación (particularmente S10 y EXA). De esta forma, estos cuatro gestores de memoria son peores que Kingsley y GEA en todas las condiciones.

Respecto al uso de memoria (Figura 6.10(b)) se puede observar que Kingsley no es el mejor gestor, en gran medida porque tiene un nivel de fragmentación interna alto que se traduce en una mayor utilización de memoria. Aunque Lea está altamente optimizado en el uso de memoria, se puede ver que en *soplex* no trabaja bien. Esto sucede porque *soplex* utiliza casi todos los tamaños de bloque en el rango [1 ..... 3849000] bytes. En este caso, Lea utiliza tamaños de bloque grandes y se produce fragmentación interna. Una vez más, el DMM obtenido con GEA es la mejor opción: 31,04 %, 4,58 %, 3,72 %, 13,17 %, 14,05 %, 20,80 %, 52,44 %, 2,05 %, 7,65 %, y 8,14 % mejor que Kingsley en *hmmer*, *dealII*, *soplex*, *calculix*, *gcc (200, epxr2, c-typeck)*, y *perlbench (split, diff, spam)*, respectivamente. Como conclusión, GEA es la mejor opción.

En este punto, se observa que el uso del simulador DMM propuesto en el proceso de exploración de memoria permite el estudio de seis mecanismos de asignación diferentes, a partir de una única ejecución previa de cada aplicación objetivo. Este aspecto junto a la tarea de perfilado de las aplicaciones y las características del simulador, hacen posible un gran ahorro en la exploración del proceso de optimización de memoria dinámica, proporcionando grandes prestaciones a los diseñadores.

Una vez analizados los resultados numéricos usando la metodología propuesta, a continuación se describe la estructura de los mejores DMM a medida, obtenidos por GEA para las 10 configuraciones abordadas.

hmmer

En la Tabla 6.7 se describe el DMM obtenido por GEA y GEA$*$ para la aplicación *hmmer*. Esta aplicación trabaja con 2013 tamaños de bloque diferentes en el rango 2 ... 9696 bytes (9,47 kb). El DMM personalizado incluye dos gestores

internos. El primero es una lista diferenciada (*segregated free list* que permite unir bloques para obtener el tamaño solicitado. Tiene una única lista de bloques libres que cubre el rango (2 ... 9600] bytes (especificado en la última columna de la tabla) y se implementa con una estructura *B-tree* con un mecanismo de asignación del primer ajuste (*first-fit*) y la política primero en entrar, primero en salir (*FIFO*). El segundo gestor sigue un ajuste óptimo diferenciado (*exact segregated fit*) con *splitting* y *coalescing*. Contiene una lista de bloques libres implementada con un árbol binario, con políticas *first fit* y último en entrar, primero en salir (*last-in-first-out*) y sus tamaños varían de 9,37 a 9,47 KB. Dado que no hay tamaños de bloque comprendidos entre 9600 y 9696 bytes, es lógico tener un gestor óptimo para almacenar bloques de tamaño máximo. En este sentido, el rendimiento es cercano al del gestor Kingsley (con sólo 2 estructuras de datos) y la memoria se gestiona mejor.

Tabla 6.7: DMM personalizado mapeado para la aplicación *hmmer*.

Estructura de datos	Mecanismo(Política)	Rango (bytes)
SegregatedFreeList, split=false, coalesce=true		
BTREE	FIRST(FIFO)	(0,9600]
ExactSegregatedFit, split=true, coalesce=true		
BTREE	FIRST(LIFO)	(9600,9696]

dealII

En la Tabla 6.8 se muestra el DMM obtenido para la aplicación *dealII*. El DMM incluye 419 tamaños diferentes de bloque que van de 4 a 7832240 bytes (7, 4 MB). El DMM obtenido por GEA presenta 4 gestores internos distintos, tres de ellos con *splitting* y *coalescing*. El primero es un gestor *binary buddy*, donde todas las listas de bloques libres se implementan como listas simplemente enlazadas con mecanismo de ajuste óptimo y política *LIFO*, asignando tamaños de bloque en el rango de 0 a 8192 bytes. El segundo gestor sigue una lista de bloques libres diferenciada y tiene una única lista de bloques libres, implementada como un árbol binario con algoritmo de asignación *first-fit* y política *LIFO* que cubre el rango 8 – 64 KB. A este gestor le sigue uno *binary buddy* con sólo una lista de bloques libres, implementa-

da como una lista simplemente enlazada con mecanismos de asignación *first-fit* y política FIFO. En este caso, el 96 % de las operaciones *malloc* se concentran en el primer gestor que permite que el DMM personalizado obtenga un rendimiento similar a Kingsley. Adicionalmente, la mayoría de los tamaños de bloque son potencia de dos. De esta forma, el algoritmo de ajuste óptimo es una buena opción para el primer y el último gestor.

Tabla 6.8: DMM personalizado mapeado para la aplicación *dealII*.

Estructura de datos	Mecanismo(Política)	Rango (bytes)
BuddySystemBinary, split=false, coalesce=false		
SLL	EXACT(LIFO)	(0,1]
SLL	EXACT(LIFO)	(1,2]
SLL	EXACT(LIFO)	(2,4]
	...	
SLL	EXACT(LIFO)	(4096,8192]
SegregatedFitList, split=true, coalesce=true		
BTREE	FIRST(LIFO)	(8192,65536]
BuddySystemBinary, split=true, coalesce=true		
BTREE	EXACT(FIFO)	(131072,262144]
BTREE	EXACT(FIFO)	(262144,524288]
BTREE	EXACT(FIFO)	(524288,1048576]
	...	
BTREE	EXACT(FIFO)	(4194304,8388608]

soplex

Con idéntico formato a las anteriores, la Tabla 6.9 presenta el DMM que GEA y GEA$*$ obtienen para la aplicación *soplex*. Esta aplicación utiliza 47728 tamaños diferentes de bloque comprendidos entre 1 y 3849000 bytes ($3,67$ MB). El DMM diseñado a medida es casi igual al gestor Kingsley, pero con *splitting* y *coalescing* activados, lo cual permite ahorrar memoria. En este caso, el uso de memoria (en términos de tamaños de bloque) se distribuye uniformemente en el rango entero. Esta diversificación no permite al algoritmo de optimización encontrar patrones en los que un gestor a medida mejoraría la fragmentación interna.

Tabla 6.9: DMM personalizado mapeado para la aplicación *soplex*.

Estructura de datos	Mecanismo(Política)	Rango (bytes)
BuddySystemBinary, split=true, coalesce=true		
DLL	FIRST(FIFO)	(0,4194304]

calculix

La Tabla 6.10 describe el DMM obtenido para *calculix*. *Calculix* utiliza 939 tamaños diferentes de bloques que varían entre 4 y 8214296 bytes (7, 83 MB). El DMM lo componen 3 gestores internos. El primero es un gestor *binary buddy* que permite *splitting* y *coalescing*. Incluye 17 listas individualmente enlazadas y todas ellas aplican el mecanismo de asignación *first-fit* y política *FIFO*. Los otros dos gestores siguen un mecanismo de lista de bloques libres diferenciada y ambas contienen una única lista. La primera lista se implementa como un árbol binario con mecanismo de asignación *first-fit* y política *FIFO*, y la segunda como una lista doblemente enlazada con mecanismo *exact-fit* y política *FIFO*. El 99, 46 % de los bloques solicitados se concentran en el primer gestor. Debido a que el primero es *binary buddy*, el rendimiento se aproxima al de Kingsley. Los otros dos gestores están optimizados para el 0, 54 % de bloques restantes. Esto permite al DMM personalizado ahorrar más de un 13, 17 % del uso de la memoria con respecto a Kingsley (ver Tabla 6.6).

Tabla 6.10: DMM personalizado mapeado para la aplicación *calculix*.

Estructura de datos	Mecanismo(Política)	Rango (bytes)
BuddySystemBinary, split=true, coalesce=true		
SLL	EXACT(LIFO)	(0,1]
SLL	EXACT(LIFO)	(1,2]
SLL	EXACT(LIFO)	(2,4]
	...	
SLL	EXACT(LIFO)	(32768,65536]
SegregatedFreeList, split=false, coalesce=false		
BTREE	FIRST(FIFO)	(65536,177240]
SegregatedFreeList, split=false, coalesce=false		
BTREE	EXACT(FIFO)	(177240,8214296]

gcc (200, expr2, c-typeck)

En la Tabla 6.11 se describen los tres DMM obtenidos por GEA para las tres configuraciones de la aplicación *gcc*. El número de tamaños de bloque utilizado en cada caso ha sido de 13964, 5957, y 8177 para *gcc/200*, *gcc/expr2* y *gcc/c-typeck*, respectivamente. Los DMM obtenidos para *gcc/200* y *gcc/expr2* son casi iguales (dos gestores *binary buddy*, el segundo con *splitting* y *coalescing* activados), mientras que el último DMM es parcialmente distinto, siendo el segundo gestor una lista de bloques libres diferenciada. Sin embargo, también se encuentra un gestor *binary buddy* como el segundo gestor entre los cinco mejores DMM para gcc/c-typeck (en las 10 ejecuciones), con un ahorro del $43,13\,\%$ del uso de memoria en vez de un $52,44\,\%$ con respecto al DMM Kingsley (ver Tabla 6.6). Así, la metodología propuesta ha sido capaz de obtener un DMM personalizado similar con dos *binary buddy* para *gcc*, independientemente de la entrada seleccionada.

Tabla 6.11: DMM personalizado mapeado para la aplicación *gcc*.

gcc/200		
Estructura de datos	Mecanismo(Política)	Rango (bytes)
BuddySystemBinary, split=false, coalesce=false		
DLL	FIRST(FIFO)	(0,1]
DLL	FIRST(FIFO)	(1,2]
DLL	FIRST(FIFO)	(2,4]
	...	
DLL	FIRST(FIFO)	(8K,16K]
BuddySystemBinary, split=true, coalesce=true		
BTREE	FIRST(FIFO)	(16K,32K]
	...	
BTREE	FIRST(FIFO)	(8M,16M]
gcc/expr2		
BuddySystemBinary, split=false, coalesce=false		
DLL	FIRST(FIFO)	(0,1]
DLL	FIRST(FIFO)	(1,2]
DLL	FIRST(FIFO)	(2,4]
	...	
DLL	FIRST(FIFO)	(8K,16K]
BuddySystemBinary, split=true, coalesce=true		
BTREE	FIRST(FIFO)	(16K,32K]
	...	
BTREE	BEST(FIFO)	(128M,256M]
gcc/c-typeck		
BuddySystemBinary, split=false, coalesce=false		
DLL	FIRST(FIFO)	(0,1]
DLL	FIRST(FIFO)	(1,2]
DLL	FIRST(FIFO)	(2,4]
	...	
DLL	FIRST(FIFO)	(64K,128K]
SegregatedFreeList, split=false, coalesce=true		
BTREE	BEST(FIFO)	(128M,256M]

perlbench (split, diff, spam)

Por último, en la Tabla 6.12 se muestran los tres DMM obtenidos para la aplicación perl. Esta aplicación se ha configurado con tres entradas diferentes para buscar

la posibilidad de obtener el mismo DMM para las tres configuraciones. En este caso, como muestra la Tabla 6.12, los tres DMM son completamente diferentes. Si se examinan las tres trazas, se observa que el patrón en la peticiones para las tres configuraciones difieren en gran medida. Se encuentran 3397, 555 y 1863 tamaños diferentes de bloque para *perl/split*, *perl/diff* y *perl/spam*, respectivamente. El tamaño de bloque más solicitado ha sido 105 KB, 9 KB, y 1 KB en cada caso, que da una idea de la naturaleza distinta de las tres configuraciones. Por lo que se refiere a las trazas, *perl/split* comienza con tamaños de bloque mayor, mientras que *perl/-diff* y *perl/spam* se concentran más en tamaños pequeños de bloque. De esta forma, GEA no encuentra un DMM óptimo para las tres configuraciones.

Como conclusión, GEA es capaz de encontrar DMM óptimos para cada una de las seis aplicaciones. En el caso de *gcc*, GEA es capaz de encontrar un DMM común para configuraciones diferentes. Sin embargo, en el caso de *perl* no ha sido posible, debido a la naturaleza diferente de esta aplicación cuyo comportamiento depende en gran medida de la entrada dada. Adicionalmente, todos los DMM personalizados comienzan con un gestor *binary buddy* que concentra más del 90 % de los tamaños de bloque, intentando aproximar el rendimiento al DMM Kingsley mientras mejora la gestión de memoria en el 10 % restante. En general, es una buena política, desde que obtiene una mejora del 33, 35 % en el valor objetivo global.

Tabla 6.12: DMM personalizado mapeado para la aplicación *perl*.

perl/split		
Estructura de datos	Mecanismo(Política)	Rango (bytes)
BuddySystemBinary, split=false, coalesce=false		
DLL	EXACT(FIFO)	(0,1]
DLL	EXACT(FIFO)	(1,2]
	...	
DLL	EXACT(FIFO)	(128K,256K]
SegregatedFreeList, split=true, coalesce=true		
BTREE	BEST(FIFO)	(256K,1M]
perl/diff		
BuddySystemBinary, split=false, coalesce=false		
DLL	FIRST(FIFO)	(0,1]
DLL	FIRST(FIFO)	(1,2]
	...	
DLL	FIRST(FIFO)	(64,128]
BuddyFibonacci, split=false, coalesce=false		
SLL	FIRST(FIFO)	(128,144]
SLL	FIRST(FIFO)	(144,233]
SLL	FIRST(FIFO)	(233,377]
	...	
BTREE	BEST(FIFO)	(610,987]
SegregatedFreeList, split=false, coalesce=true		
BTREE	BEST(LIFO)	(987,497K]
perl/spam		
BuddySystemBinary, split=false, coalesce=false		
SLL	FIRST(FIFO)	(0,1]
SLL	FIRST(FIFO)	(1,2]
	...	
SLL	FIRST(FIFO)	(256,512]
SegregatedFreeList, split=true, coalesce=true		
BTREE	BEST(FIFO)	(512,8K]
SimpleSegregatedStorage, split=false, coalesce=false		
BTREE	BEST(FIFO)	(8K,128K]

6.3.4. Conclusiones

En esta tesis, se propone un nuevo método de optimización multi-objetivo basado en GE que utiliza perfilado estático de las aplicaciones para realizar implementaciones de DMM compuestos para aplicaciones con requisitos altos de memoria dinámica. Este método simplifica el esfuerzo a realizar por los diseñadores para explorar DMM multi-capa y permite el perfeccionamiento de implementaciones de DMM de forma automática. Como resultado, el enfoque propuesto conduce a un importante ahorro del tiempo total de integración de sistemas para las aplicaciones dinámicas. Adicionalmente, el método obtiene implementaciones óptimas de estructuras DMM con respecto a las principales métricas que utilizan los diseñadores. Además, los resultados con seis aplicaciones y cinco DMM de propósito general muestran que el enfoque de optimización presentado reduce de forma significativa el tiempo de ejecución y el uso de memoria más de un $59,27\,\%$ en promedio, comparada con el valor objetivo global.

En resumen, este análisis proporciona un marco útil no intrusivo para evaluar y diseñar un DMM a medida de acuerdo a una aplicación objetivo. Así, se proporciona un enfoque simplificado para que el diseñador del sistema operativo realice la toma de decisión de seleccionar o desarrollar el algoritmo DMM a implementar en el kernel, dependiendo de las restricciones del escenario de ejecución.

Capítulo 7

Conclusiones y Trabajo Futuro

7.1. Conclusiones

Los nuevos dispositivos multimedia han incrementado su capacidad y han ampliado su funcionalidad incorporando aplicaciones más complejas. Estas aplicaciones incluyen requisitos de memoria muy altos que hacen que se incremente la presión sobre los diferentes niveles del subsistema de memoria (por ejemplo, registros del procesador, memoria cache y memoria principal), y por tanto, este deba ser optimizado. Este hecho lleva a que los investigadores centren su trabajo en mejorar el tiempo de ejecución, el consumo de energía, o ambos, así como a abordar la gestión de la memoria dinámica para realizar una asignación eficiente. Para mejorar el tiempo de ejecución, los sistemas empotrados actuales incluyen una jerarquía de memoria cache parecida a la suministrada para los ordenadores de sobremesa. Sin embargo, el amplio rango de posibles configuraciones de cache complica la búsqueda de la mejor configuración de cache. Sin mencionar el hecho que tiempo de ejecución y consumo de energía se ven afectados directamente por los patrones de acceso a memoria de las aplicaciones. En consecuencia, a lo largo de los años se han propuesto nuevas y eficientes técnicas para explorar y decidir la mejor configuración de cache para las diferentes aplicaciones. Sin embargo, no hemos encontrado en la literatura ningún trabajo que optimice, al mismo tiempo, todos los parámetros que configuran tanto la memoria cache de instrucciones como la de datos, para un conjunto de aplicaciones dado.

Por otra parte, el gestor de memoria dinámica es un componente fundamental

que influye no sólo en el comportamiento general de la aplicación sino también en el comportamiento hardware. Para gestionar eficientemente la memoria dinámica en estas aplicaciones, los ingenieros de software a menudo escriben gestores de memoria personalizados a partir de cero, lo cual es un proceso difícil y propenso a errores. De esta forma, se hace necesario diseñar metodologías que permitan, por otra parte dar respuesta a la pregunta clave de cómo medir de forma ágil la influencia de un DMM en el comportamiento de la plataforma hardware y de las aplicaciones, permitiendo diseñar gestores de memoria a medida de las aplicaciones que van a ser soportadas por estos dispositivos.

En esta tesis se ha llevado a cabo un estudio de los diferentes niveles de la jerarquía de memoria. En primer lugar se ha analizado el impacto térmico del banco de registros debido al alto número de accesos que soporta y su influencia sobre la temperatura del procesador. El objetivo es la reducción del impacto térmico y, en consecuencia, el consumo de energía. Para ello se ha presentado un marco de optimización basado en computación evolutiva, perfilado estático de aplicaciones y análisis de la temperatura en estado estacionario. Se ha abordado el problema en el banco de registros de las arquitecturas VLIW y ARM. El proceso de optimización devuelve una permutación espacial de los accesos a los registros del banco de registros, que aleja aquellos con una mayor densidad de potencia y, por consiguiente, mayor temperatura, con el fin de facilitar la disipación de calor.

El proceso de optimización utiliza un MOEA, concretamente NSGA-II bi-objetivo para evaluar de forma independiente cada una de las aplicaciones objetivo. Se ha seleccionado un subconjunto de las aplicaciones Mediabench representativas de las aplicaciones multimedia. Después de recibir la traza de la aplicación correspondiente, el proceso de optimización evalúa las soluciones en cuanto a su viabilidad física y el impacto térmico sobre el banco de registros y finalmente obtiene un conjunto de soluciones que representan una distribución espacial del banco de registros.

Los resultados obtenidos muestran que el incremento de temperatura para ambas arquitecturas es bajo, es decir, el impacto térmico en el banco de registros no es

significativo y por tanto, el proceso de optimización no sería necesario. Una de las razones es que las optimizaciones realizadas por el compilador también influyen en la reducción del impacto térmico. Sin embargo, se ha considerado que algunos de los valores obtenidos eran dignos de estudio y se ha procedido al proceso de optimización.

Los resultados de la optimización han demostrado que es posible reducir, aun más, el impacto térmico. Para todas las aplicaciones abordadas, las soluciones aportadas por el algoritmo multiobjetivo consiguen reducir el incremento de temperatura del banco de registros en todas y cada una de las tres configuraciones estructurales y arquitecturas propuestas.

En segundo lugar, se ha abordado la optimización de la memoria cache poniendo el foco de atención en las operaciones que se realizan sobre ella. Es decir, se ha considerado el rendimiento y el consumo de energía, ambos relativos a las operaciones de memoria cache. Utilizando como referencia una memoria cache con un único nivel, dividida en cache de instrucciones y de datos separadas, se han presentado tres marcos de optimización basados en computación evolutiva y perfilado estático de aplicaciones. El objetivo es mejorar el diseño de la memoria cache para sistemas empotrados, especialmente en dispositivos móviles. Con este fin se ha diseñado un marco de optimización con tres algoritmos evolutivos distintos. El marco de optimización se divide en tres fases, dos de ellas comunes para cada enfoque evolutivo y la tercera dependiente del algoritmo evolutivo. A continuación se resumen las principales conclusiones obtenidas para cada uno de ellos.

- Proceso de optimización con GE: se ha presentado un enfoque basado en Gramáticas Evolutivas como metaheurística que permite la búsqueda de la mejor configuración de cache para un conjunto de aplicaciones y un sistema objetivo. Se ha seleccionado la familia de procesadores ARM9, ampliamente utilizada en dispositivos móviles y empotrados y por la flexibilidad en el diseño de la memoria cache. De los parámetros que definen una configuración de cache, se han considerado 11: 5 parámetros que definen la memoria

cache de instrucciones y 6 para la de datos. El proceso se ha dividido en dos fases, una fase *off-line* que se realiza sólo una vez y que es responsable de la obtención de las trazas de las aplicaciones y de la caracterización de las diferentes configuraciones hardware de las memorias caches. La segunda fase tiene como finalidad conducir el proceso de optimización aplicando GE y Dinero IV, como simulador de cache para obtener la configuración que mejore el comportamiento de la aplicación objetivo.

La primera conclusión es que el marco de optimización presentado es lo suficientemente flexible como para adaptarse a cualquier cambio en el número de parámetros a evaluar o a cambios en el simulador de cache a utilizar. Ello se debe a que la gramática definida para GE facilita la comunicación con Dinero IV dado que el fenotipo se forma mediante los parámetros y formato requeridos para la llamada al simulador de cache. De esta forma, simplemente se tendría que modificar la gramática. Además, si el simulador de cache permitiera parámetros de longitud variable, una gramática apropiada sería capaz de generar la llamada correcta al simulador sin ninguna modificación en el marco de optimización.

Respecto a los resultados experimentales, se ha probado con 12 aplicaciones pertenecientes al conjunto de aplicaciones Mediabench considerando una configuración cache actual como configuración base que actúa como punto de referencia. Los resultados muestran que incluso con un número de generaciones y un tamaño de la población bajos, GE encuentra soluciones donde la función objetivo mejora más del 62 %. Dado que tiempo de ejecución y energía forman parte de la función objetivo, se ha realizado el estudio particular de ambos que presentan una mejora promedio del 87, 18 % en el consumo de energía y 37, 75 % en tiempo de ejecución frente a la configuración base de cache.

Por último, se ha realizado el análisis tiempo de ejecución del proceso de optimización que permite almacenar las soluciones ya evaluadas, evitando tener

que evaluarlas si aparecen de nuevo y reduciendo el tiempo de ejecución. Los resultados muestran un ahorro del 93, 6 % frente a una implementación sin dicho almacenamiento. Por tanto, la implementación con GE es capaz de realizar las ejecuciones en un tiempo razonable debido a que memoriza aquellos individuos que ya han sido evaluados y de obtener un conjunto de configuraciones de cache que mejoran el tiempo de ejecución y el consumo de energía de las aplicaciones objetivo.

- Proceso de optimización multiobjetivo (NSGA-II): se ha diseñado un proceso dividido en tres fases, donde la primera tiene como objetivo la obtención de las trazas de programa; la segunda es responsable de la caracterización física de las memorias cache y la tercera es dirigida por NSGA-II como algoritmo evolutivo multiobjetivo y Dinero IV, responsables de evaluar cada aplicación bajo el conjunto de configuraciones candidatas de cache.

  El resultado de la optimización es un conjunto de configuraciones de cache que reducen el tiempo de ejecución y/o el consumo de energía para cada aplicación. La reducción de ambos objetivos permite mejorar el rendimiento e incrementar la vida útil tanto de la batería como de los dispositivos. Los resultados se comparan con aquellos obtenidos por una cache comúnmente utilizada en dispositivos móviles multimedia como configuración base. Los resultados experimentales muestran una mejora promedio 64, 43 % y 91, 69 % en tiempo de ejecución y consumo de energía, respectivamente.

  Para validar los resultados, se añaden dos nuevas configuraciones de base de cache correspondientes a dispositivos con arquitectura Cortex-A9 (iPad 2, iPod-touch o iApple-TV) y Cortex-A15 (iPhone 5, iPhone 5s). Los resultados obtenidos presentan una mejora para la primera de ellas de 24, 15 % y 88, 90 % y para la segunda de 16, 85 % y 89, 84 % en tiempo de ejecución y consumo de energía, respectivamente.

  Posteriormente, se modifica el espacio de búsqueda y la configuración base a

los utilizados en el enfoque con GE y los resultados obtenidos alcanzan una mejora promedio de $33,87\,\%$ y $71,79\,\%$ en tiempo de ejecución y consumo de energía respectivamente.

Se observa que respecto al objetivo de consumo de energía para las tres primeras configuraciones propuestas el porcentaje de mejora es similar y es aproximadamente del $90\,\%$ y en la última si bien se reduce al $71,79\,\%$, este es un porcentaje de mejora excelente. Sin embargo, se produce una reducción considerable en el porcentaje de mejora en tiempo de ejecución que se sitúa del $64\,\%$ al $24\,\%$, $17\,\%$ y $33,87\,\%$ para la configuraciones base 2, 3 y 4, respectivamente. Esto se explica por la naturaleza específica de la primera configuración, mientras que las 2 siguientes son para dispositivos de propósito general, con una gran variedad tamaños de bloque. El resultado es un nivel de asociatividad más bajo que reduce el tiempo de ejecución y consume más energía. Explicación extrapolable a la configuración común a GE y NSGA-II, que para el mismo tamaño de cache que la primera presenta un tamaño de bloque de $32B$, en lugar de $16B$ y una asociatividad que pasa de 64 a 4 vías. Si bien el incremento del tamaño de bloque de forma aislada podría llevar a un incremento del tiempo de ejecución, combinado con la modificación de otros parámetros como la asociatividad en este caso, lleva a reducir el tiempo de ejecución y el consumo de energía.

Finalmente, aunque se hace necesaria la decisión humana para seleccionar la mejor memoria cache a implementar, proceso que no resulta complicado de realizar, se ha comprobado que esta metodología es capaz de encontrar un conjunto de configuraciones que mejoran el tiempo de ejecución y el consumo de energía, frente a una configuración base proporcionada como punto de referencia.

- Proceso de optimización con DE: se ha desarrollado un enfoque de optimización dirigido por DE para la búsqueda de la mejor configuración de cache

para una serie de aplicaciones objetivo de forma conjunta. El enfoque sigue la metodología general dividida en tres fases, la primera fase se encarga de la caracterización de cache, la segunda fase de suministrar las trazas de programa y la tercera aplica el algoritmo DE para explorar el espacio de búsqueda de las diferentes configuraciones de cache. Los parámetros a optimizar son los parámetros arquitectónicos clásicos, junto a los algoritmos de reemplazo, búsqueda tanto para la cache de instrucciones como de datos, y la política de escritura para la memoria cache de datos completan los 11 parámetros considerados en esta propuesta.

Las pruebas experimentales se han realizado sobre un subconjunto de 6 aplicaciones pertenecientes al conjunto Mediabench y los resultados se han comparado con una configuración de cache como punto de referencia. Los resultados obtenidos muestran que el marco propuesto es capaz de encontrar, al menos, una configuración de cache que mejora más del $62,5\,\%$ la configuración de cache que actúa como referencia. Además, este porcentaje de mejora es compartido por varias configuraciones de cache y un $22\,\%$ de las configuraciones de cache evaluadas alcanzan un porcentaje de mejora superior al $50\,\%$.

A la vista de los resultados, se observa que los parámetros arquitectónicos típicos de cache son los más influyentes. Además, el comportamiento de la cache de datos varía más que el comportamiento de la cache de instrucciones, según los parámetros de cache. Por lo tanto, algunos parámetros de cache, particularmente para la cache de instrucciones, podrían no tenerse en cuenta con el fin de reducir el tiempo necesario del proceso de optimización.

Además, aunque se considera necesario simplificar el marco de optimización, este es capaz de encontrar varias soluciones que mejoran todas las aplicaciones objetivo, siendo un excelente resultado.

En resumen, el marco de optimización de la memoria cache utilizando diferen-

tes metaheurísticas proporciona una buena herramienta al diseñador del sistema a la hora de diseñar una memoria cache, según las aplicaciones a ejecutar sobre dicho sistema. La principal ventaja de GE sobre el resto es la versatilidad que representa a la hora de cambiar los parámetros (tanto en tipo como en número) y el simulador de cache, si fuera necesario. Sin embargo, la explotación del espacio de búsqueda se realiza de forma más eficiente por parte de la optimización multiobjetivo mediante NSGA-II, tal y como se ha observado al comparar los resultados alcanzados. Además teniendo en cuenta que tiempo de ejecución y consumo de energía son objetivos en conflicto, la optimización multiobjetivo es la que mejor encaja en la resolución de estos problemas. Respecto a la propuesta con DE, dado que el marco de optimización conlleva la simulación de cada una de las configuraciones de cache para todas y cada una de las aplicaciones objetivos, se ha seleccionado DE por su sencillez y rapidez y buen desempeño en espacios continuos. En todos ellos se ha implementado como medida de ahorro, durante la aplicación de los algoritmos propuestos, el almacenamiento de los individuos ya evaluados. Esto evita tener que volver a evaluar un mismo individuo cada vez que aparezca en alguna de las generaciones y así, reducir considerablemente el tiempo de ejecución o RT del algoritmo.

En tercer lugar se ha abordado la memoria principal, presentando un marco de optimización multi-objetivo que utiliza un algoritmo basado en gramáticas evolutivas para diseñar gestores de memoria dinámica compuestos y optimizados de acuerdo a las características de aplicaciones altamente intensivas de memoria. Demostrada la baja influencia en cuanto a la temperatura y el consumo de energía, de acuerdo a un estudio preliminar, esta metodología utiliza el rendimiento y el uso de memoria como métricas a optimizar.

El proceso se ha probado con seis aplicaciones intensivas de memoria y los resultados se han comparado con cinco DMM de propósito general, de calidad reconocida. El objetivo es simplificar el esfuerzo que los diseñadores tienen que realizar para explorar DMM multi-capa y perfeccionar los métodos de implementación de gestores de memoria dinámica de forma automática. El enfoque propuesto es ca-

paz de alcanzar un ahorro de tiempo importante en la integración de sistemas para las aplicaciones dinámicas. Adicionalmente, el método obtiene implementaciones óptimas de estructuras DMM con respecto a las principales métricas que utilizan los diseñadores. Además, los resultados muestran que el enfoque de optimización presentado reduce de forma significativa el tiempo de ejecución y el uso de memoria más de un $59,27\,\%$ en promedio, comparada con el valor objetivo global. Desde este punto de vista, es posible concluir que GEA se ha presentado como la mejor opción en las seis aplicaciones y 10 configuraciones distintas, proporcionando mejoras del $20,08\,\%$, $2,31\,\%$, $0,17\,\%$, $6,59\,\%$, $7,21\,\%$, $10,55\,\%$, $33,35\,\%$, $1,11\,\%$, $3,61\,\%$ y $4,30\,\%$ para *hmmer*, *dealII*, *soplex*, *calculix*, *gcc* (200, *epxr2*, *c-typeck*), y *perlbench* (*split*, *diff*, *spam*), respectivamente. En cuanto al rendimiento, GEA se equipara a Kingsley obteniendo ambos el mejor resultado y, aunque en 6 de los 10 casos GEA es peor, el porcentaje diferencia es del $0,84\,\%$ respecto a Kingsley. Respecto al uso de la memoria, el DMM obtenido con GEA es la mejor opción con porcentajes de mejora del $31,04\,\%$, $4,58\,\%$, $3,72\,\%$, $13,17\,\%$, $14,05\,\%$, $20,80\,\%$, $52,44\,\%$, $2,05\,\%$, $7,65\,\%$, y $8,14\,\%$ tomando el DMM Kingsley como referencia en *hmmer*, *dealII*, *soplex*, *calculix*, *gcc (200, epxr2, c-typeck)*, y *perlbench (split, diff, spam)*, respectivamente.

Como conclusión, es importante destacar la naturaleza no intrusiva de la metodología propuesta evitando la modificación de las aplicaciones cada vez que se desea evaluar un DMM para una aplicación o tipo de aplicación determinado. El proceso de perfilado sólo debe de realizarse una vez por cada aplicación objetivo y el simulador DMM permite evaluar un DMM recibiendo la traza de la aplicación como entrada. En definitiva, este análisis proporciona un marco útil para evaluar y diseñar un DMM a medida de acuerdo a una aplicación objetivo. Así, se simplifica la tarea del diseñador del sistema operativo cuando tiene que decidir que algoritmo DMM seleccionar o implementar en el kernel, dependiendo de las restricciones del escenario de ejecución. Además, el uso de esta metodología permite obtener DMM personalizados de alta calidad dado que son comparados durante el algoritmo de op-

timización con gestores de memoria dinámica cuyo funcionamiento y calidad están demostrados.

7.2. Trabajo Futuro

Como resultado de la investigación realizada durante el desarrollo de esta tesis se pueden iniciar otros trabajos que continúen con la misma línea o abrir líneas nuevas en el futuro. A continuación se plantean algunas de las propuestas.

Automatización del proceso Los procesos de optimización diseñados necesitan que un experto decida qué configuración aplicar entre las propuestas por el algoritmo de optimización. Una herramienta que automatice todos los pasos sobre cada uno de los niveles de la jerarquía de memoria permitiría ahorrar no sólo tiempo y esfuerzo sino también costes.

Memorias cache multinivel En este trabajo de tesis se ha estudiado el primer nivel de la memoria cache, sin embargo actualmente el subsistema de memoria cache se estructura en varios niveles, memorias caches L1, L2 y L3 que trabajan a nivel del procesador y que permiten mejorar su rendimiento. Por tanto, para adaptarse a las necesidades actuales sería adecuado realizar un estudio pormenorizado para facilitar la toma de decisiones en este escenario. La propuesta englobaría la búsqueda multinivel de las mejores configuraciones a aplicar a cada nivel de la jerarquía de memoria cache que incrementen el rendimiento y reduzcan el consumo energético.

Consumo de energía en DMMs de propósito general y diseñados a medida En este trabajo de tesis se ha realizado el estudio de la influencia del DMM Lea en el consumo de energía y en la temperatura del procesador. Se propone ampliar el estudio tanto en el conjunto de aplicaciones abordadas como en el conjunto de gestores de memoria dinámica utilizados. Para ello, se debe cuantificar cuál es la influencia debida a los DMM de propósito general objeto del trabajo presentado (Kingsley,

Lea, EXA, S10, FIB) y de los DMM diseñados a medida de las aplicaciones objeto de dicho estudio.

Memorias cache orientadas a Objeto La memoria cache mejora el rendimiento al permitir un acceso más rápido a la información que contiene. Trabajar con objetos en memoria se rige por los mismos principios que trabajar con bloques de datos simples. La línea estaría orientada a profundizar en la idea de diseñar una jerarquía de memoria cache que permita almacenar y operar con objetos. Este diseño implica la adaptación de las políticas y algoritmos que permitan trabajar con objetos de la misma manera y, a la vez, que se trabaja con bloques de datos simples. De esta forma, sería posible mejorar el rendimiento de las aplicaciones.

Parte I

Appendices

Apéndice A

Publicaciones

Esta tesis ha dado origen a las siguientes publicaciones en revistas:

- Díaz Álvarez, Josefa, Risco Martín, José Luis, Colmenar Verdugo, José Manuel. Multi-objective optimization of energy consumption and execution time in a single level cache memory for embedded systems. Journal of Systems and Software. Volume 111, January 2016, Pages 200–212

- Díaz Álvarez, Josefa, Colmenar Verdugo, José Manuel, Risco Martín, José Luis, Lanchares Dávila, Juan, Garnica, Oscar. Optimizing L1 cache for embedded systems through grammatical evolution. 2015. Soft Computing. issn="1433-7479", doi="10.1007/s00500-015-1653-1", url="http://dx.doi.org/10.1007/s00500-015-1653-1"

- Risco Martín, José Luis, Colmenar Verdugo, José Manuel, Hidalgo Pérez, José Ignacio, Lanchares Dávila, Juan, Díaz Álvarez, Josefa. A methodology to automatically optimize dynamic memory managers applying grammatical evolution. Journal of Systems and Software. Volume 91, May 2014, Pages 109–123.

En congresos se han presentado los siguientes artículos:

- Optimizing Performance of L1 Cache Memory for Embedded Systems driven by Differential Evolution. Josefa Díaz Álvarez, J. Manuel Colmenar, José L. Risco-Martín, Juan Lanchares, Oscar Garnica. GECCO 2015. ACM ISBN

978-1-4503-3488-4/15/07, DOI: http://dx.doi.org/10.1145/2739482.2764635, pp. 1383-1384

- Josefa Día Álvarez, J. Manuel Colmenar, José L. Risco-Martín y Juan Lanchares. Optimización Multi-objetivo del Consumo de Energía y Tiempo de Ejecución en una Memoria Cache de primer nivel para Sistemas Empotrados. X Congreso Español sobre Metaheurísticas, Algoritmos Evolutivos y Bioinspirados (MAEB 2015), pp. 233-240.

- Josefa Díaz, J. Manuel Colmenar, José L. Risco-Martín, José L. Ayala and Oscar Garnica (2011), Quantifying the Impact of Dynamic Memory Managers into Memory-Intensive Applications, In Summer Computer Simulation Conference (SCSC 2011), pp. 160-16.

Índice de figuras

Índice de tablas

Bibliografía

[1] D. Albonesi, "Selective cache ways: on-demand cache resource allocation," in *Microarchitecture, 1999. MICRO-32. Proceedings. 32nd Annual International Symposium on*, pp. 248–259, 1999.

[2] M. Kandemir, J. Ramanujam, J. Irwin, N. Vijaykrishnan, I. Kadayif, and A. Parikh, "Dynamic management of scratch-pad memory space," in *Proceedings of the 38th Annual Design Automation Conference*, DAC '01, (New York, NY, USA), pp. 690–695, ACM, 2001.

[3] E. D. Berger, B. G. Zorn, and K. S. McKinley, "Reconsidering custom memory allocation," *SIGPLAN Not.*, vol. 37, pp. 1–12, nov 2002.

[4] M. T. Kandemir, "Reducing Energy Consumption of Multiprocessor SoC Architectures by Exploiting Memory Bank Locality," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 11, pp. 410–441, Apr. 2006.

[5] A. Varma, E. Debes, I. Kozintsev, and B. Jacob, "Instruction-level power dissipation in the intel xscale embedded microprocessor," in *In SPIEs 17th Annual Symposium on Electronic Imaging Science & Technology*, pp. 1–8, International Society for Optics and Photonics, 2005.

[6] P. Panda, N. Dutt, and A. Nicolau, "Architectural exploration and optimization of local memory in embedded systems," in *System Synthesis, 1997. Proceedings., Tenth International Symposium on*, pp. 90–97, Sep 1997.

[7] P. Panda, N. Nicolau, and A. Nicolau, "Data cache sizing for embedded processor applications," in *Design, Automation and Test in Europe, 1998., Proceedings*, pp. 925–926, Feb 1998.

[8] J. Abella and A. Gonzalez, "On reducing register pressure and energy in multiple-banked register files," in *Computer Design, 2003. Proceedings. 21st International Conference on*, pp. 14–20, Oct 2003.

[9] R. Balasubramonian, D. Albonesi, A. Buyuktosunoglu, and S. Dwarkadas, "A dynamically tunable memory hierarchy," *Computers, IEEE Transactions on*, vol. 52, pp. 1243–1258, Oct 2003.

[10] J. L. Hennessy and D. A. Patterson, *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann, 2011.

[11] D. M. Tullsen, S. J. Eggers, H. M. Levy, J. S. Emer, J. L. Lo, and R. L. Stamm, "Exploiting choice: Instruction fetch and issue on an implementable simultaneous multithreading processor," in *Proc. 23rd Int'l Sump. on Computer Architecture*, pp. 191–202, May 1996.

[12] D. M. Tullsen, S. J. Eggers, and H. M. Levy, "Simultaneous multithreading: Maximizing on-chip parallelism," in *22nd Annual International Symposium on Computer Architecture*, pp. 533–544, June 1998.

[13] S. Rusu, "ISSCC 2013: High-performance digital trends," 2013.

[14] S. Borkar and A. A. Chien, "The future of microprocessors," *Commun. ACM*, vol. 54, pp. 67–77, May 2011.

[15] C. Biber and C. Coleman, "ASIC package lid effects on temperature and lifetime," in *Semiconductor Thermal Measurement and Management Symposium (SEMI-THERM), 2011 27th Annual IEEE*, pp. 187–192, March 2011.

[16] D. Chandra, F. Guo, S. Kim, and Y. Solihin, "Predicting inter-thread cache contention on a chip multi-processor architecture," in *Proceedings of the 11th*

International Symposium on High-Performance Computer Architecture, HPCA '05, (Washington, DC, USA), pp. 340–351, IEEE Computer Society, 2005.

[17] C.-L. Shum, F. Busaba, and C. Jacobi, "IBM zEC12: The Third-Generation High-Frequency Mainframe Microprocessor," *IEEE Micro*, vol. 33, pp. 38–47, mar 2013.

[18] J.-L. Cruz, A. González, M. Valero, and N. P. Topham, "Multiple-banked Register File Architectures," in *Proceedings of the 27th Annual International Symposium on Computer Architecture*, ISCA '00, (New York, NY, USA), pp. 316–325, ACM, 2000.

[19] A. Sloss, D. Symes, and C. Wright, *ARM System Developer's Guide: Designing and Optimizing System Software*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2004.

[20] J. H. Tseng and K. Asanović, "Banked Multiported Register Files for High-frequency Superscalar Microprocessors," *SIGARCH Comput. Archit. News*, vol. 31, pp. 62–71, May 2003.

[21] P. J. Denning, "Virtual Memory," *ACM Comput. Surv.*, vol. 2, pp. 153–189, sep 1970.

[22] P. J. Denning, "The locality principle," *Commun. ACM*, vol. 48, pp. 19–24, jul 2005.

[23] H. Jo, Y. Kwon, H. Kim, E. Seo, J. Lee, and S. Maeng, "SSD-HDD-Hybrid Virtual Disk in Consolidated Environments," in *Euro-Par 2009 – Parallel Processing Workshops* (H.-X. Lin, M. Alexander, M. Forsell, A. Knüpfer, R. Prodan, L. Sousa, and A. Streit, eds.), vol. 6043 of *Lecture Notes in Computer Science*, pp. 375–384, Springer Berlin Heidelberg, 2010.

[24] B. Mao, H. Jiang, S. Wu, L. Tian, D. Feng, J. Chen, and L. Zeng, "HPDA: A Hybrid Parity-based Disk Array for Enhanced Performance and Reliability," *Trans. Storage*, vol. 8, pp. 4:1–4:20, feb 2012.

[25] T. Andre, J. Nahas, C. K. Subramanian, B. Garni, H. Lin, A. Omair, and J. Martino, W.L., "A 4-Mb 0.18- mu;m 1t1MTJ toggle MRAM with balanced three input sensing scheme and locally mirrored unidirectional write drivers," *Solid-State Circuits, IEEE Journal of*, vol. 40, pp. 301–309, Jan 2005.

[26] W. Y. Cho, B.-H. Cho, B.-G. Choi, H.-R. Oh, S. Kang, K.-S. Kim, K.-H. Kim, D.-E. Kim, C.-K. Kwak, H.-G. Byun, Y. Hwang, S. Ahn, G.-H. Koh, G. Jeong, H. Jeong, and K. Kim, "A 0.18- mu;m 3.0-V 64-Mb nonvolatile phase-transition random access memory (PRAM)," *Solid-State Circuits, IEEE Journal of*, vol. 40, pp. 293–300, Jan 2005.

[27] S. T. Hsu, T. Li, and N. Awaya, "Resistance random access memory switching mechanism," *Journal of Applied Physics*, vol. 101, no. 2, pp.–, 2007.

[28] T. Rueckes, K. Kim, E. Joselevich, G. Y. Tseng, C.-L. Cheung, and C. M. Lieber, "Carbon Nanotube-Based Nonvolatile Random Access Memory for Molecular Computing," *Science*, vol. 289, no. 5476, pp. 94–97, 2000.

[29] S. Jiang and X. Zhang, "LIRS: An Efficient Low Inter-reference Recency Set Replacement Policy to Improve Buffer Cache Performance," in *Proceedings of the 2002 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '02, (New York, NY, USA), pp. 31–42, ACM, 2002.

[30] E. Ebrahimi, C. J. Lee, O. Mutlu, and Y. N. Patt, "Prefetch-aware Shared Resource Management for Multi-core Systems," in *Proceedings of the 38th Annual International Symposium on Computer Architecture*, ISCA '11, (New York, NY, USA), pp. 141–152, ACM, 2011.

[31] Z. Wang, S. Khan, and D. Jimenez, "Improving writeback efficiency with decoupled last-write prediction," in *Computer Architecture (ISCA), 2012 39th Annual International Symposium on*, pp. 309–320, June 2012.

[32] A. S. Tanenbaum, *Modern Operating Systems*. Upper Saddle River, NJ, USA: Prentice Hall Press, 3rd ed., 2007.

[33] P. Marwedel, *Embedded System Design*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2006.

[34] E. White, *Making Embedded Systems: Design Patterns for Great Software*. O'Reilly Media, 2011.

[35] G. Buttazzo, *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications*. Real-Time Systems Series, Springer, 2011.

[36] M. Verma and P. Marwedel, *Advanced Memory Optimization Techniques for Low-Power Embedded Processors*. Springer, 2007.

[37] M. Mamidipaka and N. Dutt, "Failure mechanisms and models for semiconductor devices," tech. rep., JEDEC, 2009.

[38] B. Jacob, S. Ng, and D. Wang, *Memory Systems: Cache, DRAM, Disk*. Elsevier Science, 2010.

[39] F. Catthoor, *Ultra-Low Energy Domain-Specific Instruction-Set Processors*. Embedded Systems, Springer, 2010.

[40] A. Lambrechts, P. Raghavan, A. Leroy, G. Talavera, T. Aa, M. Jayapala, F. Catthoor, D. Verkest, G. Deconinck, H. Corporaal, F. Robert, and J. Carrabina, "Power breakdown analysis for a heterogeneous NoC platform running a video application," in *Application-Specific Systems, Architecture Processors, 2005. ASAP 2005. 16th IEEE International Conference on*, pp. 179–184, July 2005.

[41] J. L. Ayala, M. López-Vallejo, and A. Veidenbaum, "Energy-efficient register renaming in high-performance processors," *Proceedings of WASP*, pp. 56–61, 2003.

[42] K. Flautner, N. S. Kim, S. Martin, D. Blaauw, and T. Mudge, "Drowsy caches: simple techniques for reducing leakage power," in *Computer Architecture, 2002. Proceedings. 29th Annual International Symposium on*, pp. 148–157, 2002.

[43] J. L. Ayala and M. López-Vallejo, "Improving register file banking with a power-aware unroller," *Proceedings of PARC*, pp. 15–20, 2004.

[44] D. Atienza, P. Raghavan, J. Ayala, G. Micheli, F. Catthoor, D. Verkest, and M. Lopez-Vallejo, "Compiler-Driven Leakage Energy Reduction in Banked Register Files," in *Integrated Circuit and System Design. Power and Timing Modeling, Optimization and Simulation* (J. Vounckx, N. Azemard, and P. Maurine, eds.), vol. 4148 of *Lecture Notes in Computer Science*, pp. 107–116, Springer Berlin Heidelberg, 2006.

[45] J. L. Ayala, M. Lopez-Vallejo, D. Atienza, P. Raghavan, F. Catthoor, and D. Verkest, "Energy-aware compilation and hardware design for VLIW embedded systems," vol. 3, pp. 73–82, Inderscience Publishers, 2007.

[46] X. Zhou, C. Yu, and P. Petrov, "Temperature-aware Register Reallocation for Register File Power-density Minimization," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 14, pp. 26:1–26:22, Apr. 2009.

[47] W.-W. Hsieh and T. Hwang, "Thermal-aware post compilation for VLIW architectures," in *Proceedings of the 2009 Asia and South Pacific Design Automation Conference*, ASP-DAC '09, (Piscataway, NJ, USA), pp. 606–611, IEEE Press, 2009.

[48] M. M. Sabry, J. L. Ayala, and D. Atienza, "Thermal-aware Compilation for System-on-chip Processing Architectures," in *Proceedings of the 20th Sym-*

posium on Great Lakes Symposium on VLSI, GLSVLSI '10, (New York, NY, USA), pp. 221–226, ACM, 2010.

[49] S. Sharifi, A. Coskun, and T. Rosing, "Hybrid dynamic energy and thermal management in heterogeneous embedded multiprocessor SoCs," in *Design Automation Conference (ASP-DAC), 2010 15th Asia and South Pacific*, pp. 873–878, Jan 2010.

[50] D. Brooks, R. P. Dick, R. Joseph, and L. Shang, "Power, Thermal, and Reliability Modeling in Nanometer-Scale Microprocessors," *IEEE Micro*, vol. 27, pp. 49–62, May 2007.

[51] D. Brooks, P. Bose, V. Srinivasan, M. K. Gschwind, P. G. Emma, and M. G. Rosenfield, "New methodology for Early-Stage, Microarchitecture-Level Power-Performances Analysis of Microprocessors"," in *IBM J. Research and Development*, vol. 47, no. 5-6, pp. 653–670, IEEE, 2003.

[52] M. Mamidipaka and N. Dutt, "eCACTI: An enhanced power estimation model for on-chip caches," Tech. Rep. TR-04-28, CECS, UC Irvine, 2004.

[53] A. Malik, B. Moyer, and D. Cermak, "A low power unified cache architecture providing power and performance flexibility," in *Low Power Electronics and Design, 2000. ISLPED '00. Proceedings of the 2000 International Symposium on*, pp. 241–243, 2000.

[54] T. Sherwood, E. Perelman, G. Hamerly, S. Sair, and B. Calder, "Discovering and exploiting program phases," *Micro, IEEE*, vol. 23, pp. 84 – 93, nov.-dec. 2003.

[55] C. Zhang, F. Vahid, and W. Najjar, "A highly configurable cache architecture for embedded systems," in *Computer Architecture, 2003. Proceedings. 30th Annual International Symposium on*, pp. 136 – 146, june 2003.

[56] C. Zhang, F. Vahid, and W. Najjar, "Energy benefits of a configurable line size cache for embedded systems," in *VLSI, 2003. Proceedings. IEEE Computer Society Annual Symposium on*, pp. 87–91, 2003.

[57] C. Zhang, F. Vahid, and W. Najjar, "A highly configurable cache for low energy embedded systems," *ACM Trans. Embed. Comput. Syst.*, vol. 4, pp. 363–387, May 2005.

[58] W. Zang and A. Gordon-Ross, "A survey on Cache Tuning from a Power/Energy Perspective," *ACM Comput. Surv.*, vol. 45, pp. 32:1–32:49, July 2013.

[59] A. Naz, K. Kavi, M. Rezaei, and W. Li, "Making a case for split data caches for embedded applications," *SIGARCH Comput. Archit. News*, vol. 34, pp. 19–26, Sept. 2005.

[60] L. Chen, X. Zou, J. Lei, and Z. Liu, "Dynamically Reconfigurable Cache for Low-Power Embedded System," in *Natural Computation, 2007. ICNC 2007. Third International Conference on*, vol. 5, pp. 180 –184, aug. 2007.

[61] A. Gordon-Ross and F. Vahid, "A self-tuning configurable cache," in *Proceedings of the 44th annual Design Automation Conference*, DAC '07, (New York, NY, USA), pp. 234–237, ACM, 2007.

[62] A. Gordon-Ross, F. Vahid, and N. Dutt, "Fast Configurable-Cache Tuning With a Unified Second-Level Cache," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 17, pp. 80 –91, jan. 2009.

[63] S. Lopez, S. Dropsho, D. Albonesi, O. Garnica, and J. Lanchares, "Rate-Driven Control of Resizable Caches for Highly Threaded SMT Processors," in *Parallel Architecture and Compilation Techniques, 2007. PACT 2007. 16th International Conference on*, pp. 416–416, 2007.

[64] A. Gordon-Ross, J. Lau, and B. Calder, "Phase-based cache reconfiguration for a highly-configurable two-level cache hierarchy," in *Proceedings of the*

18th ACM Great Lakes symposium on VLSI, GLSVLSI '08, (New York, NY, USA), pp. 379–382, ACM, 2008.

[65] W. Wang and P. Mishra, "Dynamic Reconfiguration of Two-Level Caches in Soft Real-Time Embedded Systems," in *VLSI, 2009. ISVLSI '09. IEEE Computer Society Annual Symposium on*, pp. 145–150, 2009.

[66] W. Wang, P. Mishra, and S. Ranka, "Dynamic cache reconfiguration and partitioning for energy optimization in real-time multi-core systems," in *Proceedings of the 48th Design Automation Conference*, (New York, NY, USA), pp. 948–953, ACM, 2011.

[67] W. Wang, P. Mishra, and A. Gordon-Ross, "Dynamic cache reconfiguration for soft real-time systems," *ACM Trans. Embed. Comput. Syst.*, vol. 11, pp. 28:1–28:31, July 2012.

[68] M. Huang, M. Mehalel, R. Arvapalli, and S. He, "An energy efficient 32nm 20 MB l3 cache for Intel® Xeon® processor E5 family," in *Custom Integrated Circuits Conference (CICC), 2012 IEEE*, pp. 1–4, IEEE, 2012.

[69] D. S. Gracia, A. Ferrerón, L. M. D. Campo, T. M. Arnal, and V. V. n. Yúfera, "Revisiting LP-NUCA Energy Consumption: Cache Access Policies and Adaptive Block Dropping," *ACM Trans. Archit. Code Optim.*, vol. 11, pp. 19:1–19:26, June 2014.

[70] ARM946E-S TM, "Technical Reference Manual." http://www.arm.com/products/processors/classic/arm9/ arm946.php, 2014.

[71] A. Ghosh and T. Givargis, "Analytical design space exploration of caches for embedded systems," in *Design, Automation and Test in Europe Conference and Exhibition, 2003*, pp. 650 – 655, 2003.

[72] X. Li, T. Mitra, H. S. Negi, and A. Roychoudhury, "Design space exploration of caches using compressed traces," in *In Proceedings of the 18th annual international conference on Supercomputing*, pp. 116–125, ACM Press, 2004.

[73] A. Gordon-Ross, F. Vahid, and N. Dutt, "Fast configurable-cache tuning with a unified second-level cache," in *Proceedings of the 2005 international symposium on Low power electronics and design*, ISLPED '05, (New York, NY, USA), pp. 323–326, ACM, 2005.

[74] A. Janapsatya, A. Ignjatovic, and S. Parameswaran, "Finding optimal L1 cache configuration for embedded systems," in *Design Automation, 2006. Asia and South Pacific Conference on*, p. 6 pp., jan. 2006.

[75] D. Andrade, B. B. Fraguela, and R. Doallo, "Precise Automatable Analytical Modeling of the Cache Behavior of Codes with Indirections," *ACM Trans. Archit. Code Optim.*, vol. 4, Sept. 2007.

[76] R. Reddy and P. Petrov, "Cache Partitioning for Energy-efficient and Interference-free Embedded multitasking," *ACM Trans. Embed. Comput. Syst.*, vol. 9, pp. 16:1–16:35, Mar. 2010.

[77] T. Xingyan and D. Hongyan, "Static cache hint generation based on a profile of the OPT cache replacement," in *Computer Application and System Modeling (ICCASM), 2010 International Conference on*, vol. 9, pp. V9–84–V9–87, Oct 2010.

[78] M. Feng, C. Tian, C. Lin, and R. Gupta, "Dynamic Access Distance Driven Cache Replacement," *ACM Trans. Archit. Code Optim.*, vol. 8, pp. 14:1–14:30, Oct. 2011.

[79] A. Gordon-Ross, F. Vahid, and N. Dutt, "Combining Code Reordering and Cache Configuration," *ACM Trans. Embed. Comput. Syst.*, vol. 11, pp. 88:1–88:20, Jan. 2013.

[80] M. Palesi and T. Givargis, "Multi-objective design space exploration using genetic algorithms," in *Hardware/Software Codesign, 2002. CODES 2002. Proceedings of the Tenth International Symposium on*, pp. 67–72, 2002.

[81] A. Silva-Filho, C. Bastos-Filho, D. Falcao, F. Cordeiro, and R. Castro, "An Optimization Mechanism Intended for Two-Level Cache Hierarchy to improve energy and performance using the nsgaii algorithm," in *Computer Architecture and High Performance Computing, 2008. SBAC-PAD '08. 20th International Symposium on*, pp. 19 –26, 29 2008-nov. 1 2008.

[82] B. Bui, M. Caccamo, L. Sha, and J. Martinez, "Impact of Cache Partitioning on Multi-tasking Real Time Embedded Systems," in *Embedded and Real-Time Computing Systems and Applications, 2008. RTCSA '08. 14th IEEE International Conference on*, pp. 101–110, 2008.

[83] A. Dani, Y. Srikant, and B. Amrutur, "Efficient cache exploration method for a tiled chip multiprocessor," in *High Performance Computing (HiPC), 2012 19th International Conference on*, pp. 1–6, Dec 2012.

[84] C.-T. D. Lo, W. Srisa-an, and J. M. Chang, "The design and analysis of a quantitative simulator for dynamic memory management," *Journal of Systems and Software*, vol. 72, no. 3, pp. 443 – 453, 2004.

[85] G. Teng, K. Zheng, and W. Dong, "SDMA: A Simulation-Driven Dynamic Memory Allocator for Wireless Sensor Networks," in *Sensor Technologies and Applications, 2008. SENSORCOMM '08. Second International Conference on*, pp. 462–467, Aug 2008.

[86] C. Del Rosso, "The Method, the Tools and Rationales for Assessing Dynamic Memory Efficiency in Embedded Real-Time Systems in Practice," in *Software Engineering Advances, International Conference on*, p. 56, oct. 2006.

[87] D. Atienza, J. M. Mendias, S. Mamagkakis, D. Soudris, and F. Catthoor, "Systematic dynamic memory management design methodology for reduced

memory footprint," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 11, no. 2, pp. 465–489, 2006.

[88] D. Atienza, S. Mamagkakis, F. Poletti, J. M. Mendias, F. Catthoor, L. Benini, and D. Soudris, "Efficient system-level prototyping of power-aware dynamic memory managers for embedded systems," *Integr. VLSI J.*, vol. 39, no. 2, pp. 113–130, 2006.

[89] J. L. Risco-Martín, D. Atienza, J. Hidalgo, and J. Lanchares, "A parallel evolutionary algorithm to optimize dynamic data types in embedded systems," *Soft Computing*, vol. 12, no. 12, pp. 1157–1167, 2008.

[90] J. L. Risco-Martin, D. Atienza, R. Gonzalo, and H. J.I., "Optimization of dynamic memory managers for embedded systems using grammatical evolution.," in *In: GECCO'09: Proceedings of the 11th Annual conference on Genetic and evolutionary computation.*, p. 1609–1616, 2009.

[91] J. Risco-Martín, J. Colmenar, D. Atienza, and J. Hidalgo, "Simulation of high-performance memory allocators.," in *Proccedings of the 13th EUROMICRO Conference on Digital System Design*, p. 275–282, 2010.

[92] J. L. Risco-Martín, J. M. Colmenar, D. Atienza, and J. I. Hidalgo, "Simulation of high-performance memory allocators," *Microprocessors and Microsystems*, vol. 35, no. 8, pp. 755–765, 2011.

[93] E. D. Berger, B. G. Zorn, and K. S. McKinley, "Composing high-performance memory allocators," *SIGPLAN Not.*, vol. 36, no. 5, pp. 114–124, 2001.

[94] J. L. Risco-Martín, D. Atienza, J. M. Colmenar, and O. Garnica, "A parallel evolutionary algorithm to optimize dynamic memory managers in embedded systems," *Parallel Computing*, vol. 36, pp. 572–590, 2010.

[95] F. Glover, "Future paths for integer programming and links to artificial intelligence," *Computers and Operations Research*, vol. 13, no. 5, pp. 533 – 549, 1986. Applications of Integer Programming.

[96] H. R. Lourenço, O. C. Martin, and T. Stutzle, "Iterated local search," *arXiv preprint math/0102188*, 2001.

[97] S. Kirkpatrick, C. D. Gelatt, M. P. Vecchi, *et al.*, "Optimization by simmulated annealing," *science*, vol. 220, no. 4598, pp. 671–680, 1983.

[98] M. Dorigo and C. Blum, "Ant colony optimization theory: A survey," *Theoretical computer science*, vol. 344, no. 2, pp. 243–278, 2005.

[99] T. Back, D. B. Fogel, and Z. Michalewicz, eds., *Handbook of Evolutionary Computation*. Bristol, UK, UK: IOP Publishing Ltd., 1st ed., 1997.

[100] C. Blum and A. Roli, "Metaheuristics in combinatorial optimization: Overview and conceptual comparison," *ACM Computing Surveys (CSUR)*, vol. 35, no. 3, pp. 268–308, 2003.

[101] F. Glover and G. A. Kochenberger, *Handbook of metaheuristics*. Springer, 2003.

[102] D. B. Fogel, *Evolutionary computation: toward a new philosophy of machine intelligence*, vol. 1. John Wiley & Sons, 2006.

[103] Friedberg R. M., "A learning machine: part I," in *IBM J 2*, pp. 2–13, 1958.

[104] R. M. Friedberg, B. Dunham, and J. North, "A learning machine: part II," in *IBM J 3*, pp. 282–7, 1958.

[105] C. C. Coello, G. B. Lamont, and D. A. Van Veldhuizen, *Evolutionary algorithms for solving multi-objective problems*. Springer Science & Business Media, 2007.

[106] R. Storn and K. Price, *Differential evolution-a simple and efficient adaptive scheme for global optimization over continuous spaces*. ICSI Berkeley, 1995.

[107] L. Fogel, "Autonomous Automata Insdutr.," in *Res. 4*, pp. 14–9, 1962.

[108] L. Fogel, A. Owens, and M. Walsh, "Artificial Intelligence through Simulated Evolution," in *John Wiley & Sons, Inc.*, (New York), 1966.

[109] L. Fogel and G. Burgin, "Competitive Goal-Seeking throug Evolutionary Programing," in *Final Report under Contract no. AF19(628)-5927*, (Air Force Cambrigge Research Labs), 1969.

[110] D. Fogel, "Evolutionary computacion. toward a new philosophy of machine intelligence.," in *The Institute of Electricall and Electronic Enginers*, (New York), 1998.

[111] I. Rechenberg, "Cybernetic solution path of an experimental problem," tech. rep., Royal Air Force Establishment, 1965.

[112] H. Schwefe, "Kybernetische Evolution als Strategie der Exprimentellen Forschung in der Strömungstechnik," in *Master's thesis*, (Technical University of Berlin), 1965.

[113] I. Rechenberg, "Evolutionsstrategie: Optimierung technischer systeme nach principien der biologischen evolution," in *Frommann-Holzboog*, (Stuttgart, Alemania), 1973.

[114] H.-P. Schwefel, "Numerische Optimierung von Computer-Modellen mittels der Evolutions strategie," (Basel, Alemania), 1977.

[115] H.-P. Schwefel, "Numerical Optimization of Computer Models," (Wiley, Chichester, UK), 1981.

[116] C. A. Cuomo, C. A. Desjardins, M. A. Bakowski, J. Goldberg, A. T. Ma, J. J. Becnel, E. S. Didier, L. Fan, D. I. Heiman, J. Z. Levin, *et al.*, "Microspori-

dian genome analysis reveals evolutionary strategies for obligate intracellular growth," *Genome research*, vol. 22, no. 12, pp. 2478–2488, 2012.

[117] L. P. Steven Manos, "Optical fibre design using evolutionary strategies," vol. 21, pp. 564 – 576, 2004.

[118] T. Back, L. Kessler, and I. Heinle, "Evolutionary Strategies for Identification and Validation of Material Model Parameters for Forming Simulations," in *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation*, GECCO '11, (New York, NY, USA), pp. 1779–1786, ACM, 2011.

[119] J. Holland, "Outline for a logical theory of adaptive systems," pp. 297–314, 1962.

[120] J. Holland, *Adaptation in Natural and Artificial Systems*. University of Michigan Press, 1975.

[121] Z. Michalewicz, *Genetic Algorithms + Data Structures = Evolution Programs (3rd Ed.)*. London, UK, UK: Springer-Verlag, 1996.

[122] K. D. Jong, "Analisys of Behavior of a Class of Genetic Adaptive Systems," in *PhD Thesis*, University of Michigan, 1975.

[123] D. Goldberg, *Genetics Algorithms in Search, Optimization and Machine Learning*. Reading, Ma:Addison-Wesley, 1989.

[124] J. A. Dos Santos, C. D. Ferreira, and R. da Silva Torres, "A genetic Programming Approach for Relevance Feedback in Region-Based Image Retrieval Systems.," in *SIBGRAPI*, vol. 8, pp. 155–162, 2008.

[125] A. Khosravi, E. Mazloumi, S. Nahavandi, D. Creighton, and J. Van Lint, "A genetic algorithm-based method for improving quality of travel time prediction intervals," *Transportation Research Part C: Emerging Technologies*, vol. 19, no. 6, pp. 1364–1376, 2011.

[126] J. R. Koza and R. Poli, "Introductory Tutorials in Optimization, Search and Decision Support," 2003.

[127] F. Fernández, J. I. Hidalgo, J. Lanchares, and J. Sánchez, "A methodology for reconfigurable hardware design based upon evolutionary computation," *Microprocessors and Microsystems*, vol. 28, no. 7, pp. 363–371, 2004.

[128] Y. Azaria and M. Sipper, "GP-Gammon: Using genetic programming to evolve backgammon players," in *Genetic Programming*, pp. 132–142, Springer, 2005.

[129] M. Frade, F. F. de Vega, and C. Cotta, "Automatic evolution of programs for procedural generation of terrains for video games," *Soft Computing*, vol. 16, no. 11, pp. 1893–1914, 2012.

[130] C. Ryan, "Grammatical Evolution Tutorial," *Proceedings of Genetic and Evolutionary Computation Conference, GECCO*, 2006.

[131] A. Brabazon and M. O'Neill, *Biologically Inspired Algorithms for Financial Modelling*. Springer, 2006.

[132] E. Galván-López, J. Swafford, M. O'Neill, and A. Brabazon, "Evolving a Ms. PacMan Controller Using Grammatical Evolution," in *Applications of Evolutionary Computation* (C. Chio, S. Cagnoni, C. Cotta, M. Ebner, A. Ekrt, A. Esparcia-Alcazar, C.-K. Goh, J. Merelo, F. Neri, M. Preu, J. Togelius, and G. Yannakakis, eds.), vol. 6024 of *Lecture Notes in Computer Science*, pp. 161–170, Springer Berlin Heidelberg, 2010.

[133] D. Perez, M. Nicolau, M. O'Neill, and A. Brabazon, "Evolving Behaviour Trees for the Mario AI Competition Using Grammatical Evolution," in *Applications of Evolutionary Computation* (C. Chio, S. Cagnoni, C. Cotta, M. Ebner, A. Ekrt, A. Esparcia-Alcázar, J. Merelo, F. Neri, M. Preuss, H. Richter, J. Togelius, and G. Yannakakis, eds.), vol. 6624 of *Lecture Notes in Computer Science*, pp. 123–132, Springer Berlin Heidelberg, 2011.

[134] N. Shaker, M. Nicolau, G. Yannakakis, J. Togelius, and M. O'Neill, "Evolving levels for Super Mario Bros using grammatical evolution," in *Computational Intelligence and Games (CIG), 2012 IEEE Conference on*, pp. 304–311, 2012.

[135] J. Byrne, J. McDermott, E. Galvaón-Loópez, and M. O'Neill, "Implementing an intuitive mutation operator for interactive evolutionary 3D design," in *Evolutionary Computation (CEC), 2010 IEEE Congress on*, pp. 1–7, 2010.

[136] S. Das and P. N. Suganthan, "Differential evolution: A survey of the state-of-the-art," *Evolutionary Computation, IEEE Transactions on*, vol. 15, no. 1, pp. 4–31, 2011.

[137] K. Price, R. M. Storn, and J. A. Lampinen, *Differential Evolution: A Practical Approach to Global Optimization (Natural Computing Series)*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2005.

[138] J. Teo, "Exploring dynamic self-adaptive populations in differential evolution," *Soft Computing*, vol. 10, no. 8, pp. 673–686, 2006.

[139] J. Zhang and A. C. Sanderson, "JADE: adaptive differential evolution with optional external archive," *Evolutionary Computation, IEEE Transactions on*, vol. 13, no. 5, pp. 945–958, 2009.

[140] E. Mininno, F. Neri, F. Cupertino, and D. Naso, "Compact differential evolution," *Evolutionary Computation, IEEE Transactions on*, vol. 15, no. 1, pp. 32–54, 2011.

[141] J. D. Schaffer, "Multiple Objective Optimization with Vector Evaluated Genetic Algorithms," in *Genetic Algorithms and their Applications: Proceedings of the First International Conference on Genetic Algorithms*, (Hillsdale, New Jersey), pp. 93–100, 1985.

[142] A. Osyczka., *Multicriteria optimization for engineering design.* Academic Press, 1985.

[143] K. Deb, *Multi-Objective Optimization Using Evolutionary Algorithms*. Wiley, 2009.

[144] P. Fleming, R. Purshouse, and R. Lygoe, "Many-Objective Optimization: An Engineering Design Perspective," in *Evolutionary Multi-Criterion Optimization* (C. Coello Coello, A. Hernández Aguirre, and E. Zitzler, eds.), vol. 3410 of *Lecture Notes in Computer Science*, pp. 14–32, Springer Berlin Heidelberg, 2005.

[145] C. A. Coello Coello, "Evolutionary multi-objective optimization: a historical view of the field," *Computational Intelligence Magazine, IEEE*, vol. 1, pp. 28–36, Feb 2006.

[146] C. M. Fonseca and P. J. Fleming, "On the Performance Assessment and Comparison of Stochastic Multiobjective optimizers," in *PPSN IV: Proceedings of the 4th International Conference on Parallel Problem Solving from Nature*, (London, UK), pp. 584–593, Springer-Verlag, 1996.

[147] J. Horn, N. Nafpliotis, and D. Goldberg, "A niched Pareto genetic algorithm for multiobjective optimization," in *Evolutionary Computation, 1994. IEEE World Congress on Computational Intelligence., Proceedings of the First IEEE Conference on*, pp. 82–87 vol.1, Jun 1994.

[148] S. N. and D. K., "Multiobjective Optimization Using Nondominated Sorting in Genetic Algorithms," *Evolutionary Computation 2(3)*, pp. 221–248, 1994.

[149] E. Zitzler, M. Laumanns, and L. Thiele, "SPEA2: Improving the strength Pareto evolutionary algorithm for multiobjective optimization," in *Proceedings of the Evolutionary Methods for Design, Optimization and Control with Application to Industrial Problems*, (Barcelona, Spain), pp. 95–100, 2002.

[150] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *Evolutionary Computation, IEEE Transactions on*, vol. 6, no. 2, pp. 182–197, 2002.

[151] C. Coello, C. Dhaenens, and L. Jourdan, *Advances in Multi-Objective Nature Inspired Computing*. Studies in Computational Intelligence, Springer, 2010.

[152] E. Zitzler and L. Thiele, "Multiobjective evolutionary algorithms: a comparative case study and the strength Pareto approach," *Evolutionary Computation, IEEE Transactions on*, vol. 3, pp. 257–271, Nov 1999.

[153] E. Zitzler, L. Thiele, M. Laumanns, C. M. Fonseca, and V. G. da Fonseca, "Performance assessment of multiobjective optimizers: an analysis and review," *IEEE Transactions on Evolutionary Computation*, vol. 7, no. 2, pp. 117–132, 2003.

[154] M. P. Hansen and A. Jaszkiewicz, *Evaluating the quality of approximations to the non-dominated set*. IMM, Department of Mathematical Modelling, Technical Universityof Denmark, 1998.

[155] J. Srinivasan and S. V. Adve, "Predictive Dynamic Thermal Management for Multimedia Applications," in *Proceedings of the 17th Annual International Conference on Supercomputing*, ICS '03, (New York, NY, USA), pp. 109–120, ACM, 2003.

[156] Y. Li, B. Lee, D. Brooks, Z. Hu, and K. Skadron, "CMP design space exploration subject to physical constraints," in *High-Performance Computer Architecture, 2006. The Twelfth International Symposium on*, pp. 17–28, IEEE, 2006.

[157] F. J. Mesa-Martinez, E. K. Ardestani, and J. Renau, "Characterizing processor thermal behavior," in *ACM SIGARCH Computer Architecture News*, vol. 38, pp. 193–204, ACM, 2010.

[158] Y. A. Çengel and A. J. Ghajar, *Heat and Mass Transfer*. McGraw-Hill, 2011.

[159] Y. Yang, Z. Gu, C. Zhu, R. P. Dick, and L. Shang, "ISAC: Integrated Space-and-Time-Adaptive Chip-Package Thermal Analysis," *Trans. Comp.-Aided Des. Integ. Cir. Sys.*, vol. 26, pp. 86–99, Jan. 2007.

[160] P. Wesseling, "Introduction to Multigrid Methods," tech. rep., 1995.

[161] T. Instruments, "Texas Instruments." http://www.ti.com/lsds/ti/processors/dsp/overview.page, 2016.

[162] "Industry Developments and Models, Intelligent Systems: The Next Big Opportunity, IDC," 2011.

[163] G. Bell, "Moore's Law evolved the PC industry; Bell's Law disrupted it with players, phones, and tablets: New Platforms, tools, and sevices," *Microsoft research*, 2014.

[164] ARM Ltd., "ARM9 Processor Family." http://www.arm.com/products/processors/ classic/arm9/, 2013.

[165] H.-O. Kim, B. H. Lee, J.-T. Kim, J. Y. Choi, K.-M. Choi, and Y. Shin, "Supply switching with ground collapse for low-leakage register files in 65-nm CMOS," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 18, pp. 505–509, mar 2010.

[166] B. Middha, A. Gangwar, A. Kumar, M. Balakrishnan, and P. Ienne, "A Trimaran based framework for exploring the design space of VLIW ASIPs with coarse grain functional units," in *Proceedings of the 15th international symposium on System Synthesis*, ISSS '02, (New York, NY, USA), pp. 2–7, ACM, 2002.

[167] R. Kumar and P. Singh, "An Approach for Compiler Optimization to Exploit Instruction Level Parallelism," in *Advanced Computing, Networking and Informatics-Volume 2*, pp. 509–516, Springer, 2014.

[168] D. Burger and T.M.Austin, "The simplescalar tool set, version 2.0.," in *Technical Report*, (University of Wisconsin-Madison), pp. CS–TR–97–1342, June 1997.

[169] C. Lee, M. Potkonjak, and W. H. Mangione-Smith, "Mediabench: a tool for evaluating and synthesizing multimedia and communicatons systems," in *Proceedings of the 30th annual ACM/IEEE international symposium on Microarchitecture*, MICRO 30, (Washington, DC, USA), pp. 330–335, IEEE Computer Society, 1997.

[170] Y. Cai, M. Schmitz, A. Ejlali, B. Al-Hashimi, and S. Reddy, "Cache size selection for performance, energy and reliability of time-constrained systems," in *Design Automation, 2006. Asia and South Pacific Conference on*, pp. 6 pp.–, Jan 2006.

[171] H. Mahmood, M. Poncino, and E. Macii, "Cache Aging Reduction with Improved Performance Using Dynamically Re-sizable Cache," in *Proceedings of the Conference on Design, Automation & Test in Europe*, DATE '14, (3001 Leuven, Belgium, Belgium), pp. 174:1–174:6, European Design and Automation Association, 2014.

[172] T. Givargis, "Zero Cost Indexing for Improved Processor Cache Performance," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 11, pp. 3–25, Jan. 2006.

[173] M. O'Neill and C. Ryan, "Automatic Generation of Caching Algorithms," in *Evolutionary Algorithms in Engineering and Computer Science*, pp. 127–134, John Wiley & Sons, 1999.

[174] J. Díaz, J. I. Hidalgo, F. Fernández, O. Garnica, and S. López, "Improving smt performance: an application of genetic algorithms to configure resizable caches," in *Proceedings of the 11th Annual Conference Companion on Genetic and Evolutionary Computation Conference: Late Breaking Papers*, GECCO '09, (New York, NY, USA), pp. 2029–2034, ACM, 2009.

[175] J. Edler and M. D. Hill, "Dinero IV Trace-Driven Uniprocessor Cache Simulator." http://pages.cs.wisc.edu/ markhill/DineroIV/, 1998.

[176] CACTI, "Hp Labs." http://www.hpl.hp.com/research/cacti, 2015.

[177] J. L. Risco-Martín, D. Atienza, J. I. Hidalgo, and J. Lanchares, "Design Flow of Dynamically-Allocated Data Types in Embedded Applications Based on Elitist Evolutionary Computation Optimization," in *Proc. of the 11th Euromicro Conference on Digital System Design (DSD 2008)*, (Parma, Italy), pp. 455–463, September 2008.

[178] J. L. Risco-Martín and J. M. Colmenar, "Java Evolutionary COmputation library (JECO)."

[179] "Grammatical Evolution." http://www.grammatical-evolution.org/pubs.html, 2015.

[180] I. Dempsey, M. O'Neill, and A. Brabazon, *Foundations in Grammatical Evolution for Dynamic Environments*, vol. 194 of *Studies in Computational Intelligence*. Springer, Apr. 2009.

[181] M. O'Neill and C. Ryan, "Grammatical evolution," *IEEE Trans. Evolutionary Computation*, vol. 5, no. 4, pp. 349–358, 2001.

[182] M. O'Neill and C. Ryan, *Grammatical Evolution: Evolutionary Automatic Programming in an Arbitrary Language*. Kluwer Academic Publishers, 2003.

[183] M. O'Neill, E. Hemberg, C. Gilligan, E. Bartley, J. McDermott, and A. Brabazon, "GEVA - grammatical evolution in Java," *SIGEVOlution*, vol. 3, no. 2, pp. 17–22, 2008.

[184] MediaBench. Available at http://euler.slu.edu/ fritts/mediabench/.

[185] A. Sayyad and H. Ammar, "Pareto-optimal search-based software engineering (POSBSE): A literature survey," in *Realizing Artificial Intelligence Synergies in Software Engineering (RAISE), 2013 2nd International Workshop on*, pp. 21–27, May 2013.

[186] R. Storn and K. Price, "Differential Evolution – A Simple and Efficient Heuristic for global Optimization over Continuous Spaces," *Journal of Global Optimization*, vol. 11, no. 4, pp. 341–359, 1997.

[187] R. Core Team, "R A language and environment for statistical computing." http://www.R-project.org/, 2014.

[188] M. S. Johnstone and P. R. Wilson, "The memory fragmentation problem: solved?," in *ACM SIGPLAN Notices*, vol. 34, pp. 26–36, ACM, 1998.

[189] D. Lea, "A memory allocator." http://g.oswego.edu/dl/html/malloc.html.

[190] D. A. Barrett and B. G. Zorn, "Using lifetime predictors to improve memory allocation performance," in *ACM SIGPLAN Notices*, vol. 28, pp. 187–196, ACM, 1993.

[191] D. Grunwald and B. Zorn, "Customalloc: Efficient synthesized memory allocators," *Software: Practice and Experience*, vol. 23, no. 8, pp. 851–869, 1993.

[192] S. Meyers, "More effective C++: 35 new ways to improve your programs and designs," 1995.

[193] C.-K. Luk, R. Cohn, R. Muth, H. Patil, A. Klauser, G. Lowney, S. Wallace, V. J. Reddi, and K. Hazelwood, "Pin: building customized program analysis tools with dynamic instrumentation," in *PLDI '05: Proceedings of the 2005 ACM SIGPLAN conference on Programming language design and implementation*, (New York, NY, USA), pp. 190–200, ACM, 2005.

[194] "SPEC, Standard Performance Evaluation Corporation." Available at: http://www.spec.org., 2013.

[195] R. Jones, A. Hosking, and E. Moss, *The garbage collection handbook: the art of automatic memory management*. Chapman & Hall/CRC, 2011.

[196] P. R. Wilson, M. S. Johnstone, M. Neely, and D. Boles, "Dynamic Storage Allocation: A Survey and Critical Review," in *IWMM '95: Proceedings of the International Workshop on Memory Management*, (London, UK), pp. 1–116, Springer-Verlag, 1995.

[197] D. Atienza, S. Mamagkakis, F. Poletti, J. M. Mendias, F. Catthoor, L. Benini, and D. Soudris, "Efficient system-level prototyping of power-aware dynamic memory managers for embedded systems," *Integr. VLSI J.*, vol. 39, no. 2, pp. 113–130, 2006.

[198] J. L. Risco-Martin, J. M. Colmenar, D. Atienza, and J. I. Hidalgo, "Simulation of high-performance memory allocators," in *13th Euromicro Conference on Digital System Design*, pp. 275–282, 09/2010 2010.

[199] C. J. Risco-Martín, J.L., "PABA available at: https://code.google.com/p/paba," 2008.

[200] M. Sipser, *Introduction to the Theory of Computation*. Course Technology, 1996.

[201] D. Grunwald, B. Zorn, and R. Henderson, "Improving the Cache Locality of Memory Allocation," *SIGPLAN Not.*, vol. 28, pp. 177–186, June 1993.

[202] M. S. Johnstone and P. R. Wilson, "The Memory Fragmentation Problem: Solved," in *Proceedings of the First International Symposium on Memory Management, ACM*, Press, 1998.

[203] M. Y. Qadri and K. D. McDonald-Maier, "Data Cache-Energy and Throughput Models: Design Exploration for Embedded Processors," *EURASIP Journal on Embedded Systems*, vol. 2009, no. 725438, p. 6, 2009.